Visual Studio Team Test Quick Reference

A quick reference for users of the Team Testing features of Visual Studio Team System

Geoff Gray and the Microsoft VSTS Rangers team

3/30/2009

VSTS Rangers

This content was originally created by Geoff Gray for internal Microsoft use and then adopted and expanded as a Visual Studio Team System ("VSTS") Rangers project. "Our mission is to accelerate the adoption of Team System by delivering out of band solutions for missing features or guidance. We work closely with members of Microsoft Services to make sure that our solutions address real world blockers." -- Bijan Javidi, VSTS Rangers Lead

Summary

This document is a collection of items from public blog sites, Microsoft® internal discussion aliases (sanitized) and experiences from various Test Consultants in the Microsoft Services Labs. The idea is to provide quick reference points around various aspects of Microsoft Visual Studio® Team Test edition that may not be covered in core documentation, or may not be easily understood. The different types of information cover:

- How does this feature work under the covers?
- How can I implement a workaround for this missing feature?
- This is a known bug and here is a fix or workaround.
- How do I troubleshoot issues I am having?

The document contains two Tables of Contents (high level overview, and list of every topic covered) as well as an index. The current plan is to update the document on a regular basis as new information is found.

Revision History

- Version 2.0
  - Released 2/16/09
  - Available externally on CodePlex
  - Major reformat of document
  - Added comprehensive index

List of Chapters

List of Topics

Setup Considerations

Data

How to set up a SQL Results Database

1. Open a Visual Studio Command prompt. Type the following text:
   cd n:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE
2. In that folder, type the following text:
   SQLCMD /S localhost\sqlexpress /i loadtestresultsrepository.sql

   Optionally, you can install the database to any existing Microsoft SQL Server®. For example, if you had a SQL Server named **ContosoServer1** you would use the following command:

   Sqlcmd -s ContosoServer1 -i loadtestresultsrepository.sql

   (You might also need to use **-u** and **-p** to specify a user name and password so that you can connect to **ContosoServer1**)
3. On the **Test** menu, click **Administer Test Controllers**. The **Administer Test Controllers** dialog box is displayed.
4. In the **Load Test Results Connection String,** click the browse button (…) to display the **Connection Properties** dialog box.
5. In **Server Name**, type **localhost\sqlexpress** or the name of the server you used in step 2 such as **ContosoServer1**.
6. Under **Log on to the server**, choose **Use Windows Authentication**.
7. Under **Connect to a database**, choose **Select or enter a database name**. Select **LoadTest** from the drop-down list box.
8. Click **OK**.
9. Click **Close** in the **Administer Test Controller** dialog box.
10. For each load test you create, you will need to set the above store as your location. Do this by opening the Load Test, then highlight the **RunSettings** line and viewing the properties for it. You set the data location and type in the "Results" portion of the properties.

NOTE: After setting up your database, be sure to set the database recovery mode to "SIMPLE". If the mode is left at "FULL" then the log files will grow quite large, especially when running the cleanup commands at the end of this section.

How to Change the Location Where Agents Store Run Files

If you need to move the location that an agent uses to store the files downloaded to it for executing tests, the following steps will take care of this. On each agent machine,

- Open QTAgentService.exe.config
- Add "<add key="WorkingDirectory" value="***<location to use>***"/>" under the <appSettings> node.
- Create the ***<location to use>*** folder.

Troubleshooting

How to enable logging for test recording

The following feature was added to help debug recordings. You can create a log file of each recording which will show headers and post body as well as returned headers and response. The way to enable this is to add the following 2 keys:

```
[HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\9.0\EnterpriseTools
\QualityTools\WebLoadTest]

"CreateLog"=dword:00000001     [NOTE: 1=create; 0=do not create]

"RecorderLogFolder"="C:\\recordlogs"
```

How to enable Verbose Logging on an agent for troubleshooting

If you need to have verbose logging to debug or isolate issues with the agents including IP switching, you can turn on verbose logging in the config files.

1. Go to **c:\Program files\Microsoft Visual Studio 2008 Team Test Load Agent\LoadTest** on the agent machine.
2. Edit the **QTAgentServiceUI.exe.config** file
   a. change the EqtTraceLevel to 4
      ```
          <switches>
              <add name="EqtTraceLevel" value="4" />
      ```
   b. Change the CreateTraceListener value to yes
      ```
      <appSettings>
                <add key="CreateTraceListener" value="yes"/>
      ```

The above settings also apply to the QTAgent.exe.config, QTController.exe.config and the QTColtrollerService.exe.config files.

User Account requirements and how to troubleshoot authentication

The following information comes from a blog entry by Durgaprasad Gorti. The link at the end of this section will take you to the full article which includes a walkthrough on troubleshooting authentication issues on a test rig.

Workgroup authentication

In a Microsoft® Windows® domain environment, there is a central authority to validate credentials. In a workgroup environment, there is no such central authority. Still, we should be able to have computers in a workgroup talk to each other and authenticate users. To enable this, local accounts have a special characteristic that allows the local security authority on the computer to authenticate a "principal" in a special way.

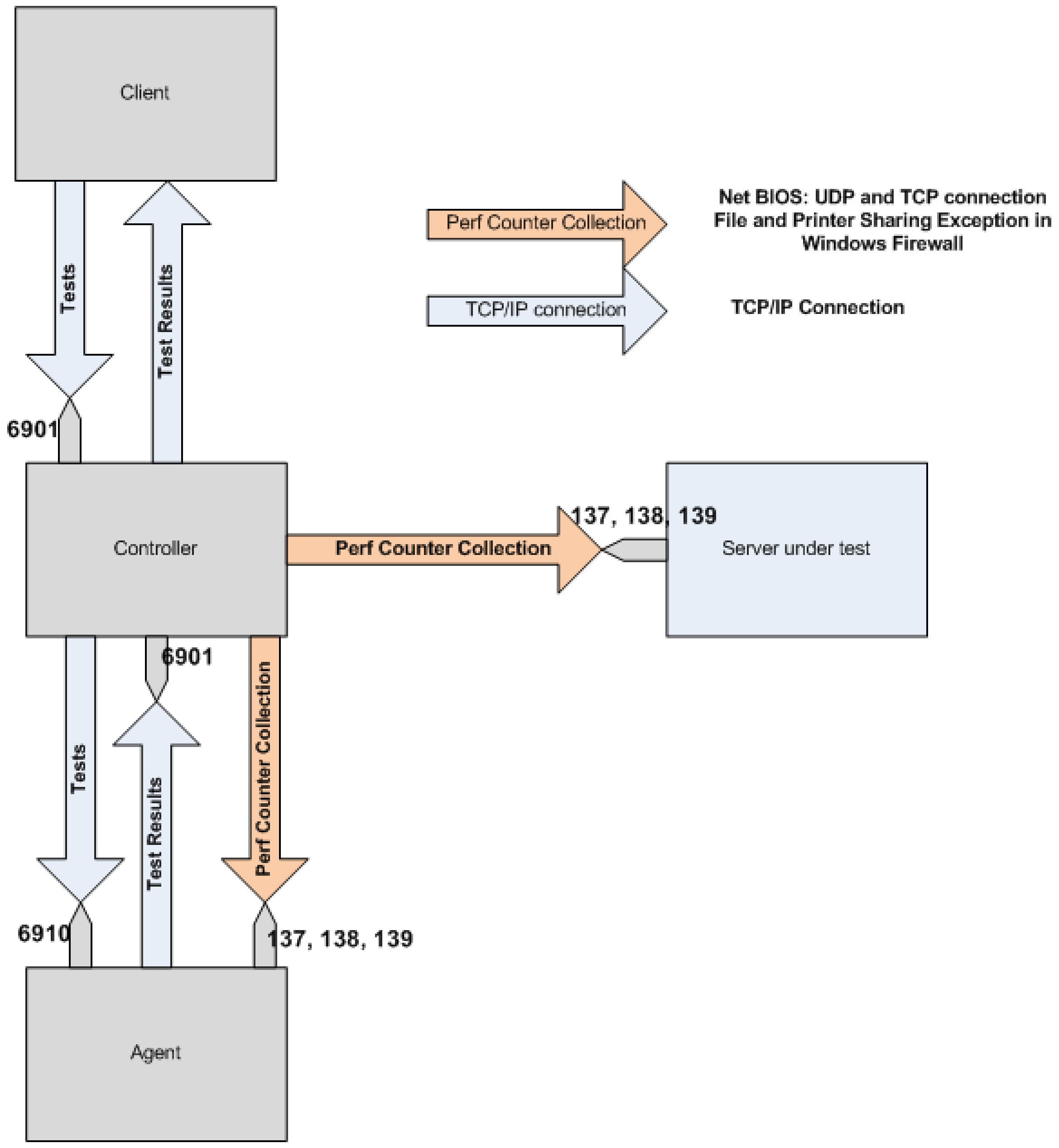
If you have two computers and a principal "UserXYZ" on both machines the security identifiers are different for MACHINE1\UserXYZ and MACHINE2\UserXYZ and for all practical purposes they are two completely different "Principals". However if the passwords are the same for them on each of these computers, the local security authority treats them as the same principal.

So when MACHINE1\UserXYZ tries to authenticate to MACHINE2\UserXYZ, and if the passwords are the same, then on MACHINE2, the UserXYZ is authenticated successfully and
is **treated as MACHINE2\UserXYZ.** Note the last sentence. The user MACHINE1\UserXYZ
is authenticated as MACHINE2\UserXYZ if the passwords are the same.

http://blogs.msdn.com/dgorti/archive/2007/10/02/vstt-controller-and-agent-setup.aspx

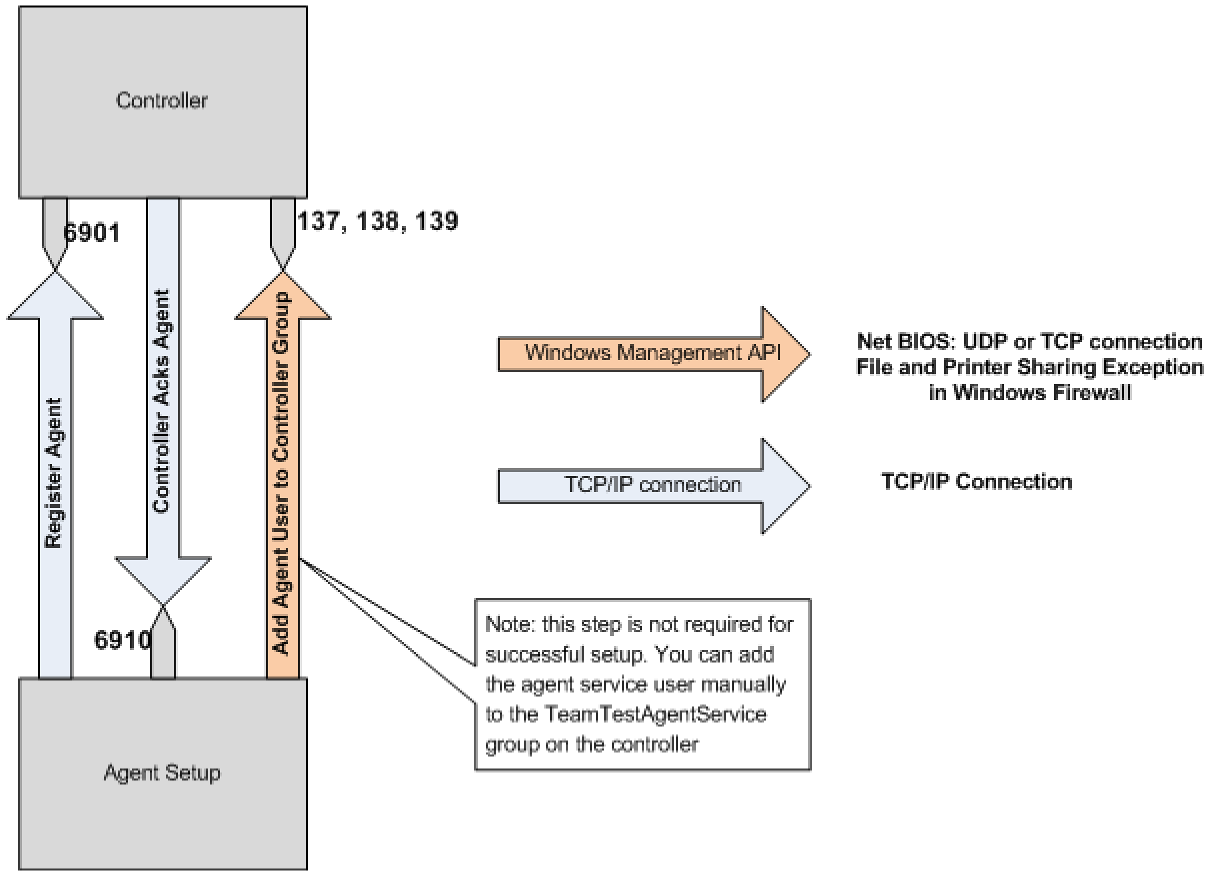
How machines in the test rig communicate

The below Visio diagrams that shows which ports are used during setup and when the agent and controller run tests.



Controller-Agent Communications

And here are the connections used during agent setup:


Controller
6901
137, 138, 139
Register Agent
Controller Acks Agent
Add Agent User to Controller Group
6910
Agent Setup
Windows Management API
Net BIOS: UDP or TCP connection File and Printer Sharing Exception in Windows Firewall
TCP/IP connection
TCP/IP Connection
Note: this step is not required for successful setup. You can add the agent service user manually to the TeamTestAgentService group on the controller

Controller-Agent Communications

Changing the Default Port for Agent-Controller Communication

The default port for communication is 6910. To change this, see the following post:

http://blogs.msdn.com/billbar/archive/2007/07/31/configuring-a-non-default-port-number-for-the-vs-team-test-controller.aspx

How to Add Agents To A Test Rig

When you uninstall the controller software and reinstall it, the local user group that contains the agent accounts used to connect is reset. You must repopulate the group with the appropriate users. From Start -> Run, type in “lusrmgr.msc” and then expand the Groups items and open the “TeamTestAgentService” group. Add the user account(s) used when setting up your agents.

Next, open VSTS and open up the Test Rig Management dialog (Test -> Administer Test Controllers) and add each agent back to the list.

Web Test Considerations

Creating Web Tests

Whitepaper on web test authoring and debugging techniques

Summary

This white paper described the new web testing features implemented in Microsoft Visual Studio® 2008. It describes how the recorder works, registry settings that will let you control what does and does not get recorded, and problems you may encounter. Finally, it introduces new debugging techniques that will enable you to find and fix your tests.

More information

http://blogs.msdn.com/edglas/archive/2007/12/02/web-test-authoring-and-debugging-techniques-for-vs-2008.aspx

Calling one coded web test from another

If you want to have two coded web tests and have one called from within the other, you need to follow a certain order to make it work:

1. Record the web tests
2. Generate code for the child
3. Include a call to the coded child in the declarative
4. Generate code for the parent

If you try to connect the two web tests before generating any code, your test will fail with the following error:

```
There is no declarative Web test with the name 'DrillDown_Coded' included in this Web
test; the string argument to IncludeWebTest must match the name specified in an
IncludeDeclarativeWebTest attribute.
```

How to use methods other than GET and POST in a web test

Summary

FormPostHttpBody and StringHttpBody are the two built-in classes for generating HTTP request bodies. If you need to generate requests containing something other than form parameters and strings then you can implement an IHttpBody class.

More information

http://blogs.msdn.com/joshch/archive/2005/08/24/455726.aspx

List of headers

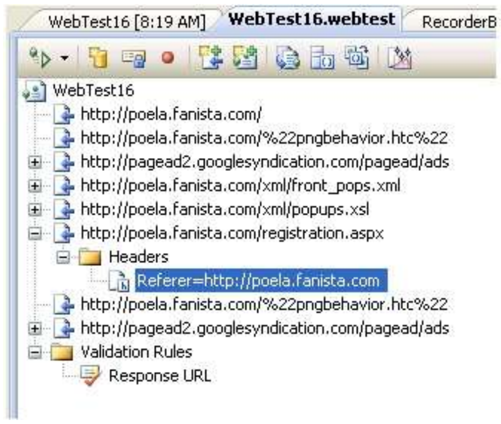
By default, the Web test recorder only records the headers:

- “SOAPAction”
- “Pragma”
- “x-microsoftajax”
- ”Content-Type”

Other headers such as the User-Agent may also be sent at runtime based on the configured browser type and any header explicitly added to the Web test request. You can change the list of headers that the Visual Studio 2008 web test recorder records in the registry by using regedit to open:

- HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\9.0\EnterpriseTools\QualityTools\Web LoadTest
- Add a string value under this key with the name "RequestHeadersToRecord” and value="SOAPAction;Pragma;x-microsoftajax;Content-Type; Referrer"

If you do this and re-record your Web test, the Referrer header should be included in the request like this:



```
WebTest16
  http://poela.fanista.com/
  http://poela.fanista.com/%22pngbehavior.htc%22
  http://pagead2.googlesyndication.com/pagead/ads
  http://poela.fanista.com/xml/front_pops.xml
  http://poela.fanista.com/xml/popups.xsl
  http://poela.fanista.com/registration.aspx
    Headers
      Referer=http://poela.fanista.com
  http://poela.fanista.com/%22pngbehavior.htc%22
  http://pagead2.googlesyndication.com/pagead/ads
  Validation Rules
    Response URL
```

Referrer header in a declarative web test

How to stop a web test in the middle of execution

If you want to stop a coded web test in the middle of execution based on a certain condition, you can hook into a couple of methods (GetRequestEnumerator or PostRequestEvent) and use the following code to stop the execution:

```
if (<condition>)
{
    this.Stop();
    yield break;
}
```

This will stop the current iteration of the test.

How to filter out certain dependant requests

Summary

One of the new Web Test features in Visual Studio 2008 is the ability to filter dependent requests. If you have a request in your web test that fetches a lot of content such as images, JavaScript files or CSS files, it's possible to programmatically determine which requests are allowed to execute during the course of the web test, and which aren't.

More information

http://blogs.msdn.com/densto/pages/new-in-orcas-filtering-dependent-requests.aspx

How to set a proxy server for web tests

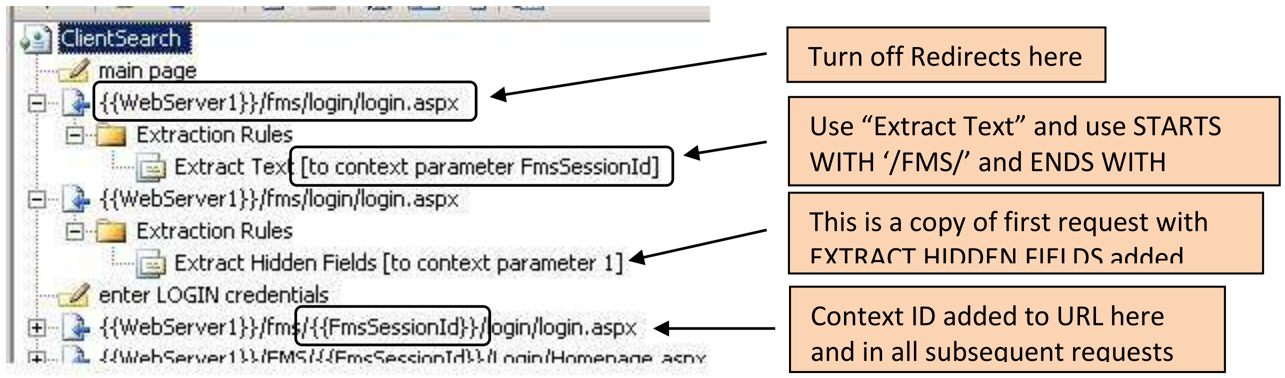
By default, there is no proxy set on a web test, so it doesn't matter what the Internet Explorer® ("IE") proxy settings are. If your test sets a specific proxy server within the web test then the IE setting is still not used. In coded web tests or web test plug-ins, you can set the proxy name using the WebProxy property of the WebTest class. **NOTE that this method is broken in Visual Studio Team Test ("VSTT") 2008 RTM, but is fixed in SP1 for VSTT 2008.**

If you wish to use the machine's IE proxy settings then you can set the Proxy property to "default" (without the quotes). In this case you should turn off Automatic Proxy Detection on each agent. Automatic Proxy detection is very slow and can greatly impact the amount of load you can drive on an agent.

Dealing with ASP.NET Cookie-less Sessions

ASP.NET allows session IDs to be passed as part of the URL requests, which can cause problems with VSTS playback. To deal with this issue, use the following steps:

1. Record the web test as normal
2. When the recording is done, you should see the very first request has no session ID in it, but all of the rest do. This first request also has a REDIRECT to the same URL, but with the session ID included.
3. For this request, turn off "Follow Redirects" and then add an extraction rule to get the value of the session (see example below).
4. Since you turned off redirects on the first request, you need to add a second request manually to the redirected page to capture any HIDDEN parameters.
5. Use "Quick Replace" to change all other hard-coded session IDs to the context you extracted in step 3



```
ClientSearch
  main page
  {{WebServer1}}/fms/login/login.aspx
    Extraction Rules
      Extract Text [to context parameter FmsSessionId]
  {{WebServer1}}/fms/login/login.aspx
    Extraction Rules
      Extract Hidden Fields [to context parameter 1]
  enter LOGIN credentials
  {{WebServer1}}/fms/{{FmsSessionId}}/login/login.aspx
  {{WebServer1}}/FMS/{{FmsSessionId}}/Login/Homepage.aspx
```

Turn off Redirects here
Use "Extract Text" and use STARTS WITH '/FMS/' and ENDS WITH
This is a copy of first request with EXTRACT HIDDEN FIELDS added
Context ID added to URL here and in all subsequent requests

How to use extracted values inside Web Request URLs

Client-side certificates and web tests

Client-side certificates are also supported in web tests, but additional code is required. The certificates need to be added to the WebTestRequest.ClientCertificates collection. This can be done in a coded web test, or by using a request plug-in in a declarative web test.

The following link describes how to use X509 certificate collections to make a SOAP request in .NET; code for using them in a web test will be similar.

More information

http://msdn.microsoft.com/en-us/library/ms819963.aspx

Removing “If-Modified-Since” header from dependent requests

The reason that If-Modified-Since headers are sent by default with dependent requests is that the web test engine attempts to emulate the behavior of Internet Explorer in its default caching mode. In many cases IE will send If-Modified-Since headers.

However, with VSTS 2008 if you want to completely disable caching of all dependent requests and always fetch them, you can so with the following WebTestPlugin:

```
    public class WebTestPlugin_DisableDependentCaching : WebTestPlugin
    {
        public override void PostRequest(object sender, PostRequestEventArgs e)
        {
            foreach (WebTestRequest dependentRequest in e.Request.DependentRequests)
            {
                dependentRequest.Cache = false;
            }
        }
    }
```

Data

CSV files created in VSTS or saved as Unicode will not work as data sources

If you create a CSV file in VSTS, it saves the file with a 2 byte prefix indicating encoding type, which is hidden. When you select the file as a data source, the first column will be prefixed with two unusual characters. The problem is the two bytes on the front that cannot be seen unless the file is viewed in hex format. The solution is to open the file in notepad and save as ANSI.

Also, if a data file is created in Windows® Notepad or Microsoft® Excel® and saved as Unicode, it looks good in Notepad or VSTS, but cannot be read in web tests. The solution is to open the file in notepad and save as ANSI.

Custom data binding in web tests

Summary

It is possible to create a custom data binding to bind to something other than a table, such as a select statement. This blog post describes one possible method – creating one class which will manage the data and creating a web test plug-in to add the data into the web test context.

More information

http://blogs.msdn.com/slumley/pages/custom-data-binding-in-web-tests.aspx

Incorrect SQL field type can cause errors in web tests

If you create a SQL table to hold test parameters and you use the default SQL column type **nchar(50)**, you will get failed requests and the context parameters in the “request” tab of the test results will not show the bad parameters. The **nchar** field pads all entries to the specified length with hidden characters but the “request” view in the test results does not show them. In order to see the extra characters, click on the “View Raw Data” checkbox and look through the data until you see the hidden characters. This will indicate that the wrong SQL field type is being used.

Adding random users to web tests

The following code can be used to generate random users for loading up sample sites with user accounts. The key to this is to randomize against a time stamp and to add another unique number (in this case, the vuser ID) so that two different instances of the load test won't accidently try to insert the same user. Issues can occur where multiple agent machines randomly generate the same user when under heavy load. The code below does not guarantee you'll never hit identical accounts, but it significantly increases the chance of never hitting it.

```
public string sRndName = "User";
public string sRndExt = @"@contoso.lab";
public int x,y;
public string sUserName;

// Generate our random user
Random randObj = new Random();
x = randObj.Next();
y = this.Context.WebTestUserId;
sUserName = sRndName + Convert.ToString(x) + Convert.ToString(y) +
sRndExt;
```

How to add a datasource value to a context parameter

If you try to assign a datasource value to a context parameter in a web test, it will not work properly. This is because VSTT does not replace datasource values in the context parameters. To work around this, you can add code directly into a coded web test or in a web test plugin. Use the following syntax for adding the binding:

```
this.Context.Add("ContextNameToUse",this.Datasource1["ColumnToUse"]);
```

Validation

New Inner-text and Select-tag rules published on Codeplex

All of the rules in this release on CodePlex relate to the inner text of a tag. For example, for a select tag (list box and combo box), the option text is stored in inner text rather than an attribute:

```
<select name="myselect1">
  <option>Milk </option>
  <option>Coffee</option>
  <option selected="selected">Tea</option>
</select>
```

In order to extract the value of the list box, we need to parse out the inner text of the selected option. TextArea is another tag that does this, but there are also a lot of other examples in HTML where you might want to extract or validate inner text. The new project has these new rules as well as a parser for inner text and select tag:

1. ExtractionRuleInnerText
2. ExtractionRuleSelectTag
3. ValidationRuleInnerText
4. ValidationRuleSelectTag

Download location

http://codeplex.com

Adding details of a validation rule to your web test

There are no properties on the WebTestResponse object or WebTestRequest object that indicate the outcome of a specific validation rule. The best approach is to have the validation rule place the result text in the WebTestContext, and then access the WebTestContext object from the WebTest object's Context property in the PostRequest or PostWebTest event handler. The following approach should work. If you have multiple validation rules, you may want to use different names for the key on the call to this.Context.Add.

```
    public class WebTest13Coded : WebTest
    {
        public WebTest13Coded()
        {
            this.PreAuthenticate = true;
        }
        public override IEnumerator<WebTestRequest> GetRequestEnumerator()
        {
            WebTestRequest request1 = new WebTestRequest("http://vsncts01/StoreCSVS");
            request1.ExpectedResponseUrl = "http://vsncts01/StoreCSVS/";
            if ((this.Context.ValidationLevel >=
Microsoft.VisualStudio.TestTools.WebTesting.ValidationLevel.High))
            {
                // request1.ValidateResponse += new
EventHandler<ValidationEventArgs>(validationRule2.Validate);
                // Specify a wrapper validation event handler …
                request1.ValidateResponse += new
EventHandler<ValidationEventArgs>(request1_ValidateResponse);
            }
            yield return request1;
            request1 = null;

            // Check the validation rule result of the previous request
            if ((bool)(this.Context["validationRule_Passed"]))
            {
                WebTestRequest request2 = new
WebTestRequest("http://vsncts01/testwebsite");
                yield return request2;
            }
        }

        private void request1_ValidateResponse(object source, ValidationEventArgs
validationEventArgs)
        {
            ValidationRuleRequiredAttributeValue validationRule = new
ValidationRuleRequiredAttributeValue();
            validationRule.TagName = "DIV";
            validationRule.AttributeName = "id";
            validationRule.MatchAttributeName = "id";
            validationRule.MatchAttributeValue = "LeftContent";
            validationRule.ExpectedValue = "LeftContent";
            validationRule.IgnoreCase = false;
            validationRule.Index = -1;

            validationRule.Validate(source, validationEventArgs);

            // Add the validation rule result to the WebTestContext
            this.Context.Add("validationRule_Passed", validationEventArgs.IsValid);
            this.Context.Add("validationRule_Message", validationEventArgs.Message);
        }
    }
```

Known Issues

Leading zeroes dropped from datasource values bound to a CSV file

If you have a datasource which contains values that start with the number 0, and you have this datasource in a CSV file, VSTT will strip the leading zero(es) from the values when using them. The same behavior does NOT occur to data values in a SQL datasource.

Web tests recorded in Fiddler may throw errors when playing back in VSTT

When Fiddler converts a trace file into a web test for VSTT, it adds every HTTP header that comes across the wire to your web test code. However, there are certain headers that should not be added by the code, since they are automatically added by the System.Net.HttpWebRequest classes. Since these are already in the core functionality, VSTT will throw a failure like:

Request failed: 100-Continue may not be set using this property.

To work around this error, simply remove the headers that should not be present in these requests. For more information, you can also visit Sean Lumley's blog post about Fiddler plugins at: http://blogs.msdn.com/slumley/pages/writing-fiddler-web-test-plugins.aspx

Recorded Think Times and paused web test recordings

When you are recording a web test, VSTS uses the time between steps as you record to generate the ThinkTime values after each request. When you add a comment, the recorder switches from RECORD mode to PAUSE mode, however, the timer to calculate think times does not pause, so you end up with think times that include the time you spent typing in the comment. This is also true if you manually pause the recording for any other reason.

WORKAROUND: Go through the test after recording is complete and adjust the think times manually.

Completion of Web Test causes spawned CMD processes to terminate

If you spawn a process from within a web test, and then that process spawns a separate CMD window (using the "START" command), the second CMD window should be totally independent of the test. If this method is used for Unit tests or for Windows applications, it will work as expected. However, web tests will kill the spawned process. Here is an excerpt from an email thread with the product team:

Here's what I've discovered. There is an option in VSTT that allows you to keep VSTestHost alive after a test run completes: go to "Tools", "Options", "Test Tools", "Test Execution" and see the check box "Keep test execution engine running between test runs". This is on by default, and I'm guessing it is on for you. When you run just a unit test in a test run, this option works and VSTestHost does not get killed when the test run completes, so neither does its child processes. However, when you run a Web test, this option seems to be ignored and VSTestHost is killed by a call to Process.Kill() which I believe does kill the child processes of VSTestHost as well (if you uncheck this option, you'll see that running the unit test has the same behavior). I'm not sure why VSTestHost goes away even when this option is set when a Web test is run – this may have been intentional. Here's a workaround that seems to work instead:

- *create a unit test that sleeps for 10 seconds (or whatever time is needed)*
- *create an ordered test that includes your coded Web test first then the unit test that sleeps*
- *run the ordered test rather than the coded Web test*

NOTE: an example of this scenario is firing off a batch file that starts a NETCAP.EXE window to gather trace data during the test run. This NETCAP process must run asynchronously so it will not block the web test. It must also complete by itself or the resultant trace file will not get written.

"Request failed: 100-Continue may not be set using this property" Error

If you generate a web test using Fiddler and run the test, you may receive errors like "…may not be set using this property".

The Fiddler plug-in that generates the .webtest adds all of the headers that Fiddler records even though some of these will generate errors when you run the Web test. The errors occur because the System.Net.HttpWebRequest (which is used by the Web test to send the request) disallows the caller from setting header that it will automatically add. The VSTS Web test recorder does filter these out, but the Fiddler recorder does not. To fix this you could either manually delete the headers in the .webtest or the generated code or you could write a Fiddler Web test plug-in to remove the header; see Sean's blog post on how to write these plug-ins at http://blogs.msdn.com/slumley/pages/writing-fiddler-web-test-plugins.

Troubleshooting

How to create Recorder Log Files (for troubleshooting Recording Issues)

To turn on the recorder log, set these registry entries:

```
[HKEY_CURRENT_USER\Software\Microsoft\VisualStudio\9.0\EnterpriseTools
\QualityTools\WebLoadTest]
```

- `"CreateLog"=dword:00000001`
- `"RecorderLogFolder"="c:\\RecorderLogs\"`

This will result in a log file for each recording session.

How to configure Web Tests so Fiddler can capture playback info

By default, web test playback ignores proxy servers set for localhost, so enabling a proxy for 127.0.0.1 (which is where Fiddler captures) will not result in any captured data. To make this work, either add a plugin with the following code, or put the following code in the Class constructor for a coded web test:

```
    this.Proxy = "http://localhost:8888";
    WebProxy webProxy = (WebProxy)this.WebProxy;
    webProxy.BypassProxyOnLocal = false;
```

Miscellaneous and "How It Works"

Exposed hooks of load and web test plugins and rules

To understand the different places you can hook into a web test or load test, and to know when they will fire, you can use "Debug.WriteLine()" in your code to output the order of execution for a test you are creating (sample output is in the next section). The places that can come (the hooks exposed) into play include:

Load Test Plugin:

```
m_loadTest.LoadTestStarting
m_loadTest.LoadTestFinished
m_loadTest.LoadTestAborted
m_loadTest.LoadTestWarmupComplete
m_loadTest.TestStarting
m_loadTest.TestFinished
m_loadTest.TestSelected
m_loadTest.ThresholdExceeded
m_loadTest.Heartbeat
```

Web Test Plugin:

```
this.PreWebTest
this.PostWebTest
this.PreTransaction (2008 SP1 and later)
this.PostTransaction (2008 SP1 and later)
this.PrePage (2008 SP1 and later)
this.PostPage (2008 SP1 and later)
```

Web Request Plugin:

```
this.PreRequest
this.PostRequest
```

Validation Rule:

```
this.ValidateResponse
```

Extraction Rule:

```
request1.ExtractValues
```

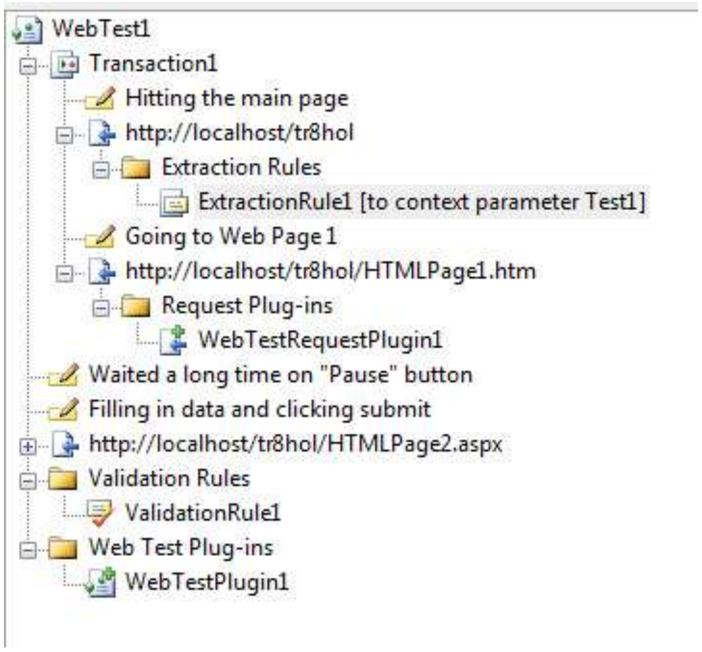
Order of execution of load and web test plugins and rules

The following is the basic order of execution for different hook points within a Web Test and its related plugins and rules. The first list is just the web test executed a single time. The second list is the load test executed (single user, 2 iterations):

```
WTPI....Web Test Plugin
CWT....Coded Web Test
ER1....Extraction Rule
VR1....Validation Rule
=====SINGLE PASS WEB TEST=====
   WTPI.... Entering constructor
   CWT...Entering constructor
   WTPI.... Entering constructor
   CWT...Entering constructor
  WTPI...PreWebTest method entry -  Iteration:1
   VR1.... Entering constructor
  TRANS.... Starting Transaction - Iteration:1
  WTPI...PreTransaction method entry - Name:Transaction1 Iteration:1
   REQUEST 1.... Entering request:1, URL:LocalHost/tr8HOL
   ER1.... Entering constructor
PrePage method entry - URL:http://localhost/tr8hol Iteration:1
  WTPI...PreRequest method entry -  Iteration:1
  WTPI...PostRequest method entry -  Iteration:1
  WTPI...PreRequest method entry -  Iteration:1
  ER1...Exiting the extraction method
    VR1....Exiting ValidationRoutine - 1
  WTPI...PostRequest method entry -  Iteration:1
  WTPI...PostPage method entry - Iteration:1
   REQUEST 1.... Exiting request:1, URL:LocalHost/tr8HOL
   REQUEST 2.... Entering request:1, URL:LocalHost/tr8HOL/HTMLPage1.htm
   WTRPI.... Entering constructor
      WTRPI....PreRequest - http://localhost/tr8hol/HTMLPage1.htm
PrePage method entry - URL:http://localhost/tr8hol/HTMLPage1.htm Iteration:1
  WTPI...PreRequest method entry -  Iteration:1
    VR1....Exiting ValidationRoutine - 1
      WTRPI....PostRequest - http://localhost/tr8hol/HTMLPage1.htm
  WTPI...PostRequest method entry -  Iteration:1
  WTPI...PostPage method entry - Iteration:1
   REQUEST 2.... Exiting request:1, URL:LocalHost/tr8HOL/HTMLPage1.htm
  WTPI...PostTransaction method entry - Name:Transaction1 Iteration:1
  TRANS.... Ending Transaction - Iteration:1
   REQUEST 3.... Entering request:1, URL:LocalHost/tr8HOL/HTMLPage2.htm
PrePage method entry - URL:http://localhost/tr8hol/HTMLPage2.aspx Iteration:1
  WTPI...PreRequest method entry -  Iteration:1
    VR1....Exiting ValidationRoutine - 1
  WTPI...PostRequest method entry -  Iteration:1
  WTPI...PostPage method entry - Iteration:1
   REQUEST 3.... Exiting request:1, URL:LocalHost/tr8HOL/HTMLPage2.htm
  WTPI...PostWebTest method entry -  Iteration:1
```

```
LTPI....Load Test Plugin
WTPI....Web Test Plugin
CWT....Coded Web Test
ER1....Extraction Rule
VR1....Validation Rule
=====SINGLE USER TWO ITERATION LOAD TEST=====
   WTPI.... Entering constructor
   CWT...Entering constructor
LTPI..Heartbeat - 0
LTPI..Test Selected - WebTest1Coded
   WTPI.... Entering constructor
   CWT...Entering constructor
LTPI..Test Starting - WebTest1Coded
  WTPI...PreWebTest method entry -  Iteration:1
  TRANS.... Starting Transaction - Iteration:1
  WTPI...PreTransaction method entry - Name:Transaction1 Iteration:1
   REQUEST 1.... Entering request:1, URL:LocalHost/tr8HOL
   ER1.... Entering constructor
PrePage method entry - URL:http://localhost/tr8hol Iteration:1
  WTPI...PreRequest method entry -  Iteration:1
  WTPI...PostRequest method entry -  Iteration:1
  WTPI...PreRequest method entry -  Iteration:1
  ER1...Exiting the extraction method
LTPI..Heartbeat - 1
  WTPI...PostRequest method entry -  Iteration:1
  WTPI...PostPage method entry - Iteration:1
   REQUEST 1.... Exiting request:1, URL:LocalHost/tr8HOL
   REQUEST 2.... Entering request:1, URL:LocalHost/tr8HOL/HTMLPage1.htm
   WTRPI.... Entering constructor
      WTRPI....PreRequest - http://localhost/tr8hol/HTMLPage1.htm
PrePage method entry - URL:http://localhost/tr8hol/HTMLPage1.htm Iteration:1
  WTPI...PreRequest method entry -  Iteration:1
      WTRPI....PostRequest - http://localhost/tr8hol/HTMLPage1.htm
  WTPI...PostRequest method entry -  Iteration:1
  WTPI...PostPage method entry - Iteration:1
   REQUEST 2.... Exiting request:1, URL:LocalHost/tr8HOL/HTMLPage1.htm
  WTPI...PostTransaction method entry - Name:Transaction1 Iteration:1
  TRANS.... Ending Transaction - Iteration:1
   REQUEST 3.... Entering request:1, URL:LocalHost/tr8HOL/HTMLPage2.htm
PrePage method entry - URL:http://localhost/tr8hol/HTMLPage2.aspx Iteration:1
  WTPI...PreRequest method entry -  Iteration:1
  WTPI...PostRequest method entry -  Iteration:1
  WTPI...PostPage method entry - Iteration:1
   REQUEST 3.... Exiting request:1, URL:LocalHost/tr8HOL/HTMLPage2.htm
  WTPI...PostWebTest method entry -  Iteration:1
LTPI..Test Finished - WebTest1Coded
LTPI..Test Selected - WebTest1Coded
   WTPI.... Entering constructor
   CWT...Entering constructor
LTPI..Test Starting - WebTest1Coded
  WTPI...PreWebTest method entry -  Iteration:2
  TRANS.... Starting Transaction - Iteration:2
  WTPI...PreTransaction method entry - Name:Transaction1 Iteration:2
   REQUEST 1.... Entering request:2, URL:LocalHost/tr8HOL
   ER1.... Entering constructor
```

```
PrePage method entry - URL:http://localhost/tr8hol Iteration:2
  WTPI...PreRequest method entry -  Iteration:2
  WTPI...PostRequest method entry -  Iteration:2
  WTPI...PreRequest method entry -  Iteration:2
  ER1...Exiting the extraction method
  WTPI...PostRequest method entry -  Iteration:2
  WTPI...PostPage method entry - Iteration:2
   REQUEST 1.... Exiting request:2, URL:LocalHost/tr8HOL
   REQUEST 2.... Entering request:2, URL:LocalHost/tr8HOL/HTMLPage1.htm
   WTRPI.... Entering constructor
      WTRPI....PreRequest - http://localhost/tr8hol/HTMLPage1.htm
PrePage method entry - URL:http://localhost/tr8hol/HTMLPage1.htm Iteration:2
  WTPI...PreRequest method entry -  Iteration:2
      WTRPI....PostRequest - http://localhost/tr8hol/HTMLPage1.htm
  WTPI...PostRequest method entry -  Iteration:2
  WTPI...PostPage method entry - Iteration:2
   REQUEST 2.... Exiting request:2, URL:LocalHost/tr8HOL/HTMLPage1.htm
  WTPI...PostTransaction method entry - Name:Transaction1 Iteration:2
  TRANS.... Ending Transaction - Iteration:2
   REQUEST 3.... Entering request:2, URL:LocalHost/tr8HOL/HTMLPage2.htm
PrePage method entry - URL:http://localhost/tr8hol/HTMLPage2.aspx Iteration:2
  WTPI...PreRequest method entry -  Iteration:2
  WTPI...PostRequest method entry -  Iteration:2
  WTPI...PostPage method entry - Iteration:2
   REQUEST 3.... Exiting request:2, URL:LocalHost/tr8HOL/HTMLPage2.htm
  WTPI...PostWebTest method entry -  Iteration:2
LTPI..Test Finished - WebTest1Coded
LTPI..Test Selected - WebTest1Coded
   WTPI.... Entering constructor
   CWT...Entering constructor
LTPI..Load Test Finished
```



```
WebTest1
  Transaction1
    Hitting the main page
    http://localhost/tr8hol
      Extraction Rules
        ExtractionRule1 [to context parameter Test1]
    Going to Web Page 1
    http://localhost/tr8hol/HTMLPage1.htm
      Request Plug-ins
        WebTestRequestPlugin1
  Waited a long time on "Pause" button
  Filling in data and clicking submit
  http://localhost/tr8hol/HTMLPage2.aspx
  Validation Rules
    ValidationRule1
  Web Test Plug-ins
    WebTestPlugin1
```

Picture of web test used to show order of execution

Understanding the Response Size reported in web test runs

If you look at the size of a response shown for a single pass of a web test (within the test results window), it may differ from the size reported from tools such as Fiddler or Netmon. This is due to the fact that VSTS is measuring the size of the response after it has been uncompressed, while Fiddler and Netmon will look at the size of the response on the wire.

This behavior has been changed in SP1, HOWEVER, there are a couple of gotchas to be aware of:

- The compressed size will only be reported in VSTS if the response in NOT using "chunked encoding"
- The test results window will not indicate whether the reported size is the compressed or the uncompressed size.
- VSTS has a receive buffer that defaults to 1,500,000 bytes and it throws away anything over that. The number reported is what is saved in the buffer, not the number of bytes received. You can increase the size of this buffer by altering the ResponseBodyCaptureLimit at the start of your test. This needs to be done in code and cannot be modified in a declarative test.

Client Code does not execute because Web Tests Work at the HTTP Layer

The following blog outlines where and how web tests work. This is important to understand if you are wondering why client side code is not tested.

http://blogs.msdn.com/slumley/pages/web-tests-work-at-the-http-layer.aspx

Proper Understanding of TRX Files and Test Results Directories

There is a test context snapshot stored before every request (including dependent requests). Sometimes, you'll find really large VIEWSTATE in a text context that can make them really large.

We added optimizations to control the amount data that is stored in the TRX for request/response bodies by only storing one copy of a unique response bodies (in mutli-iteration runs you may end up with multiple identical responses). Also, the request and response bodies are compressed to dramatically reduce the amount of space they require in the TRX. However, if I recall correctly, the XML serializer may be writing four bytes for each one byte of response (nnn, nnn, nnn where nnn is a decimal value for each byte in the compressed result).

The request/response headers and the test context snapshots are not compressed and duplicates are not eliminated, so they have the potential to become bloated.

Calls to HTTPS://Urs.Microsoft.Com show up in your script

If you record a script using IE7 and you have phishing enabled, you can get extra calls to Urs.Microsoft.Com. These calls are being made by IE as part of the phishing filter in IE (for more information, please go to: http://download.microsoft.com/download/2/8/e/28e60dcc-123c-4b27-b397-1f6b2b6cb420/Part1_MM.pdf). You may either remove these calls, or disable phishing in IE before you make the calls. To disable phishing, go to TOOLS -> PHISHING FILTER -> TURN OFF AUTOMATIC WEBSITE CHECKING.

Downloads, Download Size and Storage of files during Web Tests

The web test engine does not write responses to disk, so you don't need to specify a location for the file. It does read the entire response back to the client, but only stores the first 1.5M of the response in memory

You can override that using the `WebTestRequest.ResponseBodyCaptureLimit` property in the request's section of a coded web test.

Parameterization of HIDDEN Fields in a webtest

From an internal email thread describing how hidden fields are dealt with in VSTT:

"Hidden Field Buckets"

We call the number at the end as the bucket number, e.g. $HIDDEN0 is bucket 0.

The easiest example to explain is a frames page with two frames. Each frame will have an independent bucket, and requests can be interleaved across the frames. Other examples that require multiple buckets are popup windows and certain AJAX calls (since web tests support correlation of viewstate in ASP.NET AJAX responses).

Hidden field matching

The algorithm to determine that a given request matches a particular bucket uses the heuristic that the hidden fields parsed out of the response will match form post fields on a subsequent request.

E.g. if the recorder parses out of a response

- <INPUT type=hidden ID=Field1 value=v1>
- <INPUT type=hidden ID=Field2 value=v2>

Then on a subsequent post we see Field1 and Field2 posted, then this request and response match and a hidden field bucket will be created for them. The first available bucket number is assigned to the hidden field bucket.

Once a bucket is "consumed" by a subsequent request via binding, that bucket is made available again. So if the test has a single frame, it will always reuse bucket 0:

- Page 1
  - Extract bucket 0
- Page 2
  - Bind bucket 0 params
- Page 3
  - Extract bucket 0
- Page 4
  - Bind bucket 0 params

If a test has 2 frames that interleave requests, it will use two buckets:

- Frame 1, Page 1
  - Extract bucket 0
- Frame 2, Page 1
  - Extract bucket 1
- Frame 2, Page 2

    - Bind bucket 1 params
- Frame 1, Page 2
    - Bind bucket 0 params

Or if a test uses a popup window, or Viewstate, you would see a similar pattern as the frames page where multiple buckets are used to keep the window state.

Why are some fields unbound?

Some hidden fields values are modified in java script. I believe EVENT_ARGUMENT is example of this. In that case, it won't work to simply extract the value from the hidden field in the response and play it back. If the recorder detects this is the case, it put the actual value that was posted back as the form post parameter value rather than binding it to the hidden field.

I can't think of any case where a single page would have >1 hidden field extraction rule applied, that sounds like a bug to me. The case of multiple forms on a given page is interesting, but there should be just one down-stream post of form fields, resulting in one application of the hidden field extraction rule.

We're trying to drive as many of these out of the product as we can, if you or anyone else hits a case where this happens, send us the recording log, .webtest, and playback trx. Of course it's helpful if we can hit the website, but given these files should be enough for us to determine the problem.

Masking a 404 error on a dependent request

When running web tests, you may find that certain dependent requests always fail with a 404 error. Normally you would resolve this issue by fixing the broken link, or removing the reference. However, sometimes (for the sake of moving forward with your testing) you might want to have VSTT ignore the error. Ed Glas has a blog outlining one way to do this quickly (http://blogs.msdn.com/edglas/archive/2008/08/06/masking-a-404-error-in-a-dependent-request.aspx) but that may not work in all cases. For example if an ASPX page has some code that returns a link to a local file that is not present, then the blog post above will not work. In this case, you should consider using a plugin similar to the following (thanks to Ed Glas for the sample):

```
//*************************************************************************************
*************
// WebTestDependentFilter.cs
// Owner: Ed Glas
//
// This web test plugin filters dependents from a particular site.
// For example, if the site you are testing has ads served by another company
// you probably don't want to hit that site as part of a load test.
// This plugin enables you to filter all dependents from a particular site.
//
// Copyright(c) Microsoft Corporation, 2008
//*************************************************************************************
*************
using Microsoft.VisualStudio.TestTools.WebTesting;

namespace SampleWebTestRules
{
    public class WebTestDependentFilter : WebTestPlugin
    {
        string m_startsWith;
        public string FilterDependentRequestsThatStartWith
        {
            get { return m_startsWith; }
            set { m_startsWith = value; }
        }

        public override void PostRequest(object sender, PostRequestEventArgs e)
        {
            WebTestRequestCollection depsToRemove = new WebTestRequestCollection();

            // Note, you can't modify the collection inside a foreach, hence the
second collection
            // requests to remove.
            foreach (WebTestRequest r in e.Request.DependentRequests)
            {
                if (!string.IsNullOrEmpty(FilterDependentRequestsThatStartWith) &&
                    r.Url.StartsWith(FilterDependentRequestsThatStartWith))
                {
                    depsToRemove.Add(r);
                }
            }
            foreach (WebTestRequest r in depsToRemove)
            {
                e.Request.DependentRequests.Remove(r);
            }
        }
    }
}
```

Web Service Test Considerations

Testing Web Services with Unit Tests

If you need some help or a starting point for building Web Service tests using Unit tests, the below blog gives a great walkthrough.

http://blogs.msdn.com/slumley/pages/load-testing-web-services-with-unit-tests.aspx

Parameterizing Web Service calls within Web Tests

By default, VSTS does not expose an automated way of parameterizing the data passed in the body of a Web Service call. However, it does still honor the syntax used to define parameters in the string. To manually add a parameter definition in the body, edit the string and add the parameters where you need them. The syntax is:

{{Datasource.Table.Column}}

Here is a sample:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <OrderItem xmlns="http://tempuri.org/">
      <userName>jb@ibuyspy.com</userName>
      <password>IBS_007</password>
      <productID>{{DataSource1.Products.ProductID}}</productID>
      <quantity>1</quantity>
    </OrderItem>
  </soap:Body>
</soap:Envelope>
```

Unit Test Considerations

How to Do Things

How to add think time to a Unit Test

When you use a web test, the VSTS environment provides a property for each request called ThinkTime. This is the preferred method to use. However, there is no such property for Unit Tests. In order to simulate think time within Unit Tests, use the Windows API "Sleep" and pass in the appropriate value (the parameter for sleep is in milliseconds, so use 1000 to simulate 1 second of sleep time). The Sleep API will work well here because it is a non-CPU intensive API. The reason it is NOT recommended for web tests is because it is a blocking API and more than one web test can share a thread, therefore it can adversely affect more than one vuser. Unit tests do not share threads, therefore they are not affected by this

Passing Load Test Context Parameters to Unit Tests

http://blogs.msdn.com/slumley/archive/2006/05/15/passing-load-test-context

Global Variables in a Unit Test

If you need to have a global variable shared among iterations of a unit test, use the following:

Define a static member variable of the unit test class, or if you have multiple unit test classes that need to share the data, create a singleton object that is accessed by all of the unit tests.   The only case in which this would not work is if you have multiple unit test assemblies being used in the same load test that all need to share the global data and you also need to set the "Run Unit Tests in Application Domain" load test setting to true.   In that case each unit test assembly has its own app domain and its own copy of the static or singleton object.

CAVEAT: This will not work in a multi-agent test rig. If you have a multi-agent rig and you want truly global data, you'd either need to create a common Web service or use a database that all of the agents access.

Custom Data Binding in UNIT Tests

The first thing to do is create a custom class that does the data initialization (as described in the first part of this post: http://blogs.msdn.com/slumley/pages/custom-data-binding-in-web-tests.aspx). Next, instantiate the class inside your unit test as follows:

```
[TestClass]
    public class VSTSClass1
    {
        private TestContext testContextInstance;

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

        [ClassInitialize]
        public static void ClassSetup(TestContext a)
        {
            string m_ConnectionString = @"Provider=SQLOLEDB.1;Data 
Source=dbserver;Integrated Security=SSPI;Initial Catalog=Northwind";
            CustomDs.Instance.Initialize(m_ConnectionString);
        }
        [TestMethod]
        public void Test1()
        {
            Dictionary<string, string> dictionary = customDs.Instance.GetNextRow();
            //......Add the rest of your code here.
        }
```

Using Unit Tests to Drive Load with Command Line Apps

The following code can be used in a Unit Test to drive a command line tool (such as a testing tool). The Unit test can then be driven by a load test to emulate multiple copies of the app.

```
using System.Threading;
using System.Diagnostics;
using System.IO;
.......
[TestMethod]
 public void TestMethod1()
 {
     int x=0;
     int iDuration = 10000;

     try
     {
         Process myProcess = new Process();
         myProcess = Process.Start("c:\\temp\\conapp2.exe", “arg1”, “arg2”);

         myProcess.WaitForExit(iDuration); //Max iDuration milliseconds to return

         if (!myProcess.HasExited) //If the app has not exited, kill it manually
         {
             myProcess.Kill();
             Console.WriteLine("Application hung and was killed manually.");
         }
         else
         {
             x = myProcess.ExitCode;
             Console.WriteLine("Completed. Exit Code was {0}", x);
         }
     }
     catch (Exception e)
     {
         Console.WriteLine("The following exception was raised: " + e.Message);
     }
     finally
     {
     }
 }
```

Known Issues

Possible DESKTOP HEAP errors when driving command line unit tests

When you run a large number of unit tests that call command line apps, and they are run on a test rig (this does not happen when running tests locally), you could have several of the tests fail due to running out of desktop heap. You need to increase the amount of heap that is allocated to a service and decrease the amount allocated to the interactive user. See the following post for in depth information, and consider changing the registry as listed below:
http://blogs.msdn.com/ntdebugging/archive/2007/01/04/desktop-heap-overview.aspx

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems
OLD SETTING: "Windows SharedSection=1024,3072,512"
NEW SETTING: "Windows SharedSection=1024,1024,2460"
```

Miscellaneous and "How It Works"

Unit Tests that require the use of App.Config files (App Domain Issue)

When a unit test is run by itself, a separate application domain is created in the test process for each unit test assembly. There is some overhead associated with marshalling tests and test results across the application domain boundary, so when running unit tests in a load test, the application domain is not created by default. This provides some performance boost in terms of the number of tests per second that the test process can execute before running out of CPU. The only drawback is that if the unit test depends on an app.config file, this doesn't work without creating the app domain. In this case, you can enable the creation of app domain for the unit tests: in the Load Test editor's Run Setting's properties set the property "Run unit tests in application domain" to True.

Adding Console Output to the results store when running Unit tests under load

The following link points to a write-up on how to allow unit tests to write custom output messages to the Load Test Results Store database from Unit tests while they are running in a load test:

http://blogs.msdn.com/billbar/pages/adding-console-output-to-load-tests-running-unit-tests.aspx

Testing execution order in Unit Tests

I think that most confusion comes from some user's expectation of MSTest to execute like the Nunit framework. They execute differently since Nunit instantiates a test class only once when executing all the tests contained in it, whereas MSTest instantiates each test method's class separately during the execution process, with each instantiation occurring on a separate thread. This design affects 3 specific things which often confuse users of MSTest:

1. **ClassInitialize and ClassCleanup**: Since ClassInitialize and ClassCleanUp are static, they are only executed once even though several instances of a test class can be created by MSTest. ClassInitialize executes in the instance of the test class corresponding to the first test method in the test class. Similarly, MSTest executes ClassCleanUp in the instance of the test class corresponding to the last test method in the test class.
2. **Execution Interleaving**: Since each instance of the test class is instantiated separately on a different thread, there are no guarantees regarding the order of execution of unit tests in a single class, or across classes. The execution of tests may be interleaved across classes, and potentially even assemblies, depending on how you chose to execute your tests. The key thing here is – all tests could be executed in any order, it is totally undefined.
3. **TextContext Instances**: TestContexts are different for each test method, with no sharing between test methods.

For example, if we have a Test Class:

```
[TestClass]
    public class VSTSClass1
    {
        private TestContext testContextInstance;

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

        [ClassInitialize]
        public static void ClassSetup(TestContext a)
        {
            Console.WriteLine("Class Setup");
        }
        [TestInitialize]
        public void TestInit()
        {
            Console.WriteLine("Test Init");

        }
       [TestMethod]
        public void Test1()
        {
            Console.WriteLine("Test1");
        }
        [TestMethod]
        public void Test2()
        {
            Console.WriteLine("Test2");

        }
        [TestMethod]
        public void Test3()
        {
            Console.WriteLine("Test3");
        }
        [TestCleanup]
        public void TestCleanUp()
        {
            Console.WriteLine("TestCleanUp");
        }
        [ClassCleanup]
        public static void ClassCleanUp ()
        {
            Console.WriteLine("ClassCleanUp");
        }
    }
```

(This consists of 3 Test Methods, ClassInitialize, ClassCleanup, TestInitialize, TestCleanUp and an explicit declaration of TestContext)

The execution order would be as follows:

Test1 [Thread 1]: new TestContext -> ClassInitialize -> TestInitialize -> TestMethod1 -> TestCleanUp
Test2 [Thread 2]: new TestContext -> TestInitialize -> TestMethod2 -> TestCleanUp
Test3 [Thread 3]: new TestContext -> TestInitialize -> TestMethod2 -> TestCleanUp -> ClassCleanUp

The output after running all the tests in the class would be:

```
Class Setup
Test Init
Test1
TestCleanUp
Test Init
Test2
TestCleanUp
Test Init
Test3
TestCleanUp
ClassCleanUp
```

Load Test Considerations

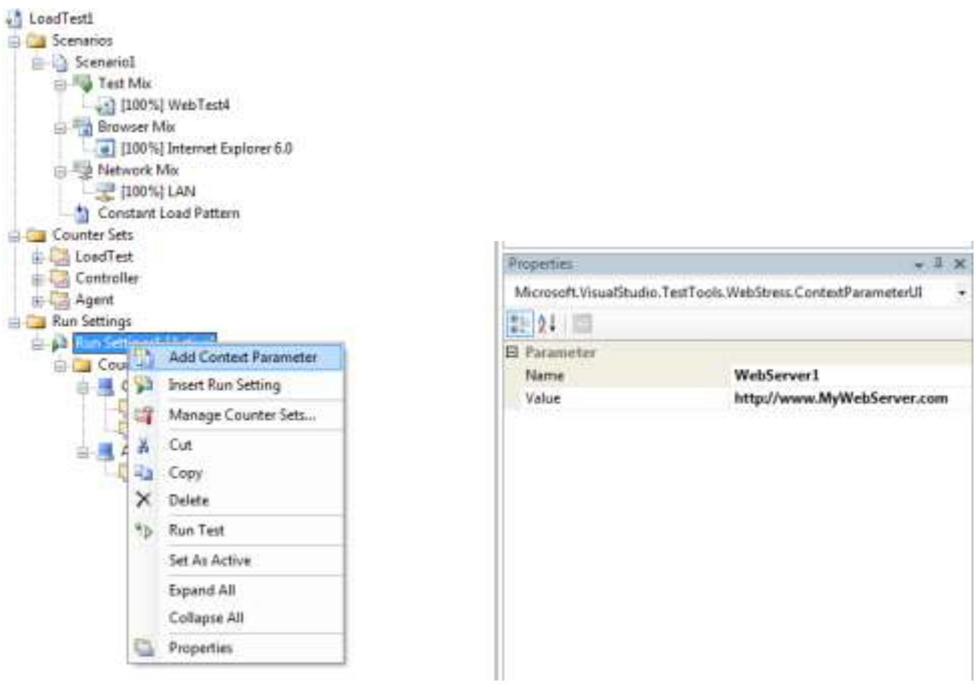
Settings available for load tests

Using the “Test Iterations” Setting

In the properties for the Run Settings of a load test, there is a property called “Test Iterations” that tells VSTS how many tests iterations to run during a load test. This is a global setting, so if you choose to run 5 iterations and you have 10 vusers, you will get FIVE total passes, not fifty. NOTE: you must enable this setting by changing the property “Use Test Iterations” from FALSE (default) to TRUE.

Adding parameters to Load Tests

To add a parameter to a Load Test, open the load test and right-click on the “Run Settings1” line (or wherever you want to add the parameter) and then choose to add a context parameter. Make sure it uses the same name as the parameter you wish to override in the web tests if that is your intent.



Name	WebServer1
Value	http://www.MyWebServer.com

Adding parameters to load tests

How to Change the Standard Deviation for a NormalDistribution ThinkTime

Find the <test_name>.loadtest file in the VSTT project directory and edit it directly. You will find a section like the one below for each scenario in the loadtest. Change the ThinkProfile Value to whatever standard deviation you wish to use. The default value in VSTT is 20% (0.2)

```
<Scenario Name="Scenario1" DelayBetweenIterations="2"
PercentNewUsers="0" IPSwitching="true"
TestMixType="PercentageOfTestsStarted">
      <ThinkProfile Value="0.2" Pattern="NormalDistribution" />
```

Any ThinkTime that has a value of zero will remain zero regardless of the distribution settings.

Bill Barnett Blog on various considerations for web tests running under load

The following blog entry describes a number of different features and settings to consider when running web tests under a load test in VSTT (a link to the blog entry is at the bottom of this topic). The following topics are covered:

- General Load Test Considerations
  - Verify web tests and unit tests
  - Choose an appropriate load profile
    - Using a Step Load Profile
    - Using a Goal-Based Load Profile
  - Choosing the location of the Load Test Results Store
  - Consider including Timing Details to collect percentile data
  - Consider enabling SQL Tracing
  - Don't Overload the Agent(s)
  - Add an Analysis Comment
- Consideration for Load Tests that contain Web Tests
  - Choose the Appropriate Connection Pool Model
    - ConnectionPerUser
    - ConnectionPool
  - Consider setting response time goals for web test requests
  - Consider setting timeouts for web test requests
  - Choose a value for the "Percentage of New Users" property
  - Consider setting the "ParseDependentRequests" property of your web test requests to false

http://blogs.msdn.com/billbar/articles/517081.aspx

How to access 32 bit performance counters on a 64 bit machine

If you are running ASP.NET or IIS in 32 bit mode on a 64 bit machine, remote performance counter collection does not work by default. When you try to collect these counters from a remote machine, the OS will by default try to collect 64 bit counters, not the 32 bit versions.

When remotely reading performance counters the Performance Logs and Alerts service on the remote system communicates with the Remote Registry service on the target x64 system. By default the Remote Registry service that is loaded is the 64-bit version and it looks to the 64-bit registry structure, causing the x64 system to send back the 64-bit counter information to the remote system, even if the remote system is a 32-bit OS. Thus we are unable to remote monitor any 32-bit counters on the x64 system. To change this behavior, set the following keys in the registry:

```
Reg path: (HKLM = HKEY_LOCAL_MACHINE)
HKLM\SYSTEM\CurrentControlSet\Services\RemoteRegistry
Param: ImagePath
Old Value: %SystemRoot%\System32\svchost.exe -k regsvc
New Value: %SystemRoot%\SysWow64\svchost.exe -k regsvc

Reg path: (HKLM = HKEY_LOCAL_MACHINE)
HKLM\SYSTEM\CurrentControlSet\Services\RemoteRegistry\Parameters
Param: ServiceDll
Old Value: %SystemRoot%\System32\regsvc.dll
New Value: %SystemRoot%\SysWow64\regsvc.dll
```

Copy these commands into a .reg file and run it:

```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RemoteRegistry]
"ImagePath"="%SystemRoot%\SysWow64\svchost.exe -k regsvc”
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\RemoteRegistry\Parameters]
"ServiceDll"="%SystemRoot%\SysWow64\regsvc.dll"
```

This will load the 32-bit version of the Remote Registry service, which looks to the 32-bit registry structure, causing the x64 system to send back the 32-bit counter information to the remote system.

Further testing showed that remote administration tools, like Regedit, Computer Management, System Information, etc., all still work even with the 32-bit version of Remote Registry loaded on the x64 system.

In addition to the registry changes please make sure to follow-up with a reboot of the machine where you changed the registry and also your load test controller (although it might be good enough just restarting the controller service).

Finally, the ASP.NET counter category will have the version appended to it. So you need to open <installdir>\common7\IDE\Templates\LoadTest\CounterSets\ASP.NET.counterset, and change<CounterCategory Name="ASP.NET"> To <CounterCategory Name="ASP.NET v2.0.50727">

Controlling the amount of memory that the SQL Server Results machine consumes

The default behavior for SQL Server is to consume as much memory as it thinks it can, the workload on the machine may not be allowing SQL Server to correctly identify memory pressure and hence give back some memory. You can configure SQL Server to a max memory limit, which if all you are doing is inserting results should be fine.

The below is how you can set memory to 512mb. The size of the memory you use will vary based on the machine, testing and how much memory you have.

```
sp_configure 'show advanced options', 1
RECONFIGURE
GO
sp_configure 'max server memory', 512
RECONFIGURE
GO
```

How to configure the timeouts for deployment of load tests to agents

The file to change is “Microsoft Visual Studio 9.0\Xml\Schemas\vstst.xsd”. look for the run config schema. Then search for “timeout”:

```
      <xs:element name="Timeouts" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="runTimeout" type="xs:int" use="optional"
default="0"/>
          <xs:attribute name="testTimeout" type="xs:int"
use="optional" default="1800000"/>
          <xs:attribute name="agentNotRespondingTimeout" type="xs:int"
use="optional" default="300000"/>
          <xs:attribute name="deploymentTimeout" type="xs:int"
use="optional" default="300000"/>
          <xs:attribute name="scriptTimeout" type="xs:int"
use="optional" default="300000"/>
        </xs:complexType>
      </xs:element>
```

Change the values as needed and note that the time is in milliseconds.

How to set the number of Load Test Errors and Error Details saved

There's a distinction between "Errors" and "Error Details". “Errors” refers to any type of error that occurs in the load test. "Error Details" refers to the additional detail we capture for errors on Web test requests: mostly the request and response body.

By default the load test results will save only 1000 errors of a particular type, but you can change this setting in the appropriate configuration file (depending on whether this is for local runs or for test rig runs):

If running locally, edit the `<Program Files>\Microsoft Visual Studio 9\Common7\IDE\VSTestHost.exe.config` file

If running with a controller, on the controller machine edit `<Program Files>\Microsoft Visual Studio 9.0 Team Test Load Agent\LoadTest\QTController.exe.config` file.

then add a key to the "appSettings" section of the file (add the "appSettings" section if needed) with the name "LoadTestMaxErrorsPerType" and the desired value.

For example, the following config file sets the maximum errors per type to 5000.

```
<appSettings>
   <add key="LoadTestMaxErrorsPerType" value="5000"/>
</appSettings>
```

These setting affect the total numbers of errors per each different "error subtype" and not the total number of errors.   For example with the default setting of 1000, you can get 1,000 404 Not Found errors and 1,000 "Test Errors" which are unit test failures.

Load Test behaviors

Test timeout setting for load test configuration does not affect web tests

The “Test Timeout” setting in the Test Run Configuration file (in the “Test -> Edit Test Run Configuration” menu) does not have an effect in all cases.

- **Uses the setting**
  - Running a single unit test, web test, ordered test, or generic test by itself
  - Running any of the above types of tests in a test run started from Test View, the Test List editor, or mstest.
  - Any load test that is run on a controller/agent test rig.
- **Does not use the setting**
  - Running a Web test in a load test
  - The load test itself

This particular test timeout is enforced by the agent test execution code, but load test and Web test execution are tightly coupled for performance reasons and when a load test executes a Web test, the agent test execution code that enforces the test timeout setting is bypassed.

How user pacing and “Think Time Between Tests” work

The setting “Think Time Between Tests” is available in the properties for a load test scenario. This value is applied when a user completes one test, then the think time delay is applied before the user starts the next iteration. The setting applies to each iteration of each test in the scenario mix.

If you create a load test that has a test mix model “Based on user pace”, then the pacing calculated by the test engine will override any settings you declare for “Think Time Between Tests”.

Load test warmup and cool down considerations

Warmup:

VSTS 2008 Load Test warmup periods are very quick to ramp up users (the agents are currently set to spin up 10 new users every second on each agent). If you choose to use a constant load pattern to avoid having the ramp-up requests in your final data set, you will need to implement a plugin to control the speed at which users come online during warmup. See the following post for more information and a sample code snippet:

http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=1311729&SiteId=1

NOTE: This behavior is changed in VSTT 2008 SP1. *It will now increase the number of users every 1 second by the initial user load (of the load pattern to be used after warm-up) divided by the number of seconds in the warm-up period (rounded to the nearest integer value) so that user load reaches the initial user load for the load pattern specified exactly at the end of the warm-up period*

Cool down:

The Load test Terminate method does not fire unless you use a cool down period.

Goal based user behavior after the test finishes the warmup period

1. The user load starts at the value specified by the Initial User Count property of the Goal Based Load Pattern.
2. At each sampling interval (which defaults to 5 seconds, but can be modified by the “Sample Rate” property in the load test run settings), the performance counter defined in the goal based load pattern is sampled. (If it can’t be sampled for some reason, an error is logged and the user load remains the same.)
3. The value sampled is compared with the “Low End” and “High End” properties of the “Target Range for Performance Counter”.
4. If the value is within the boundaries of the “Low End” and “High End”, the user load remains the same.
5. If the value is **not** within the boundaries of the “Low End” and “High End”, the user load is adjusted as follows:
   - The midpoint of the target range for the goal is divided by the sample valued for the goal performance counter to calculate an “adjustment factor”.
   - For example, if the goal is defined as “% Processor Time” between 50 and 70, the midpoint is 60. If the sampled value for % Processor Time is 40, then AdjustmentFactor = 60/40 = 1.5, or if the sampled value is 80, the AdjustmentFactor = 60/80 = 0.75.
   - The AdjustmentFactor is multiplied by the current user load to get the new user load.
   - However, if the difference between the new user load and the current user load is greater than the “Maximum User Count Increase/Decrease” property (whichever applies), then the user load is only adjusted by as much as max increase/decrease property. My experience has been that keeping these values fairly small is a good idea; otherwise the algorithm tends to cause too much fluctuation (the perf counter keeps going above and below the target range).
   - The new user load can also not be larger than the value specified by the goal based pattern’s MaximumUserCount property or less than the Minimum User Count property.
   - Two more considerations based on special properties of the goal based load pattern:
     - If the property “Lower Values Imply Higher Resource Use” is True (which you might use for example for a performance count such as Memory\Available Mbytes), then the user load is adjusted in the opposite direction: the user load is decreased when the sampled counter value is less than the Low End of the target range and increased when the user load is greater than the High End of the target range.
     - If the property “Stop Adjusting User Count When Goal Achieved” is True, then once the sampled goal performance counter is within the target range for 3 consecutive sampling intervals, then the user load is no longer adjusted and remains constant for the remainder of the load test.
   - Lastly, as is true for all of the user load patterns, in a test rig with multiple agents, the new user load is distributed among the agents equally by default, or according to the “Agent Weightings” if these are specified in the agent properties.

What is the difference between Unique, Sequential and Random Data Sources

Single Machine running tests

Sequential – This is the default and tells the web test to start with the first row then fetch rows in order from the data source. When it reaches the end of the data source, loop back to the beginning and start again. Continue until the load test completes.

Random – This indicates to choose rows at random. Continue until the load test completes.

Unique – This indicates to start with the first row and fetch rows in order. Once every row is used, stop the web test. If this is the only web test in the load test, then the load test will stop.

Multiple machines running as a rig

Sequential – This works that same as if you are on one machine. Each agent receives a full copy of the data and each starts with row 1 in the data source. Then each agent will run through each row in the data source and continue looping until the load test completes.

Random – This also works the same as if you run the test on one machine. Each agent will receive a full copy of the data source and randomly select rows.

Unique – This one works a little differently. Each row in the data source will be used once. So if you have 3 agents, the data will be spread across the 3 agents and no row will be used more than once. As with one machine, once every row is used, the web test will stop executing.

Threading models in Unit tests under load

There needs to be one thread for each virtual user that is currently running a test. The load test engine doesn't know what's going on inside the unit test and needs to run each on a separate thread to ensure that a thread will be available to start the next unit test without delay. However, if you specify the Test Mix Based on User Pace feature (or specify a non-zero value for "Think Time Between Test Iterations" (a property on each Scenario in the load test)), then the number of concurrent virtual users is less than the total number of virtual users, and there is only one thread needed in the thread pool for each concurrent virtual user.

There is an extra thread for each unit test execution thread that is used to monitor the execution of the unit test, implement timing out of the test, etc. However, the stack size for this thread is smaller than the default size so it should take up less memory.

More information can be found at: http://blogs.msdn.com/billbar/pages/features-and-behavior-of-load-tests-containing-unit-tests-in-vsts-2008.aspx

Simulation of Browser Caching during load tests

In a VSTS load test that contains Web tests, the load test attempts to simulate the caching behavior of the browser. Here are some notes on how that is done:

- There is a property named on each request in a Web test named “Cache Control” in the Web test editor (and named “Cache” on the WebTestRequest object in the API used by coded Web tests).
- When the Cache Control property on a request in the Web test is false, the request is always issued.
- When the Cache Control property is true, the VSTS load test runtime code attempts to emulate the Internet Explorer caching behavior (with the “Automatically” setting).

(This includes reading and following the HTTP cache control directives.)

- The Cache Control property is automatically set to true for all dependent requests (typically for images, style sheets, etc embedded on the page).
- In a load test, the browser caching behavior is simulated separately for each user running in the load test.
- When a virtual user in a load test completes a Web test and a new Web test session is started to keep the user load at the same level, sometimes the load test starts simulates a “new user” with a clean cache, and sometimes the load test simulates a return user that has items cached from a previous session. This is determined by the “Percentage of New Users” property on the Scenario in the load test. The default for “Percentage of New Users” is 100 in which case all user sessions are started with a clean cache. This is probably not correct for most applications where there are return users, so users should consider the most appropriate value to use for this setting depending on the actual usage of the application being load tested.

Important Note: When running a Web test by itself (outside of the load test), the Cache Control property is automatically set to false for all dependent requests so they are always fetched; this is so that they can be displayed in the browser pane of the Web test results viewer without broken images.

Update for VSTS 2008: The Web test API enhancement in VSTS 2008 now allow you to write a WebTestPlugin that disables caching of all dependent requests. See this blog post: http://blogs.msdn.com/billbar/archive/2008/06/06/disabling-caching-of-all-dependent-requests.aspx.

To get more information after a test run, do the following:

1. Open the test results. Click Tables.
2. From the drop down, select Requests.
3. Right-click on the columns, select “Add/Remove Columns”
4. Click the Cached column. Click Ok.
5. Order the Cached column.

Troubleshooting and known issues with Load Tests

Debugging Errors in Load Tests

http://blogs.msdn.com/slumley/pages/debugging-errors-in-load-test.aspx

Debugging OutOfMemory Exceptions in Load Tests

http://blogs.msdn.com/billbar/pages/diagnosing-outofmemoryexceptions-that-occur-when-running-load-tests.aspx

LoadTestItemResults.dat file and Low Disk Space

The LoadTestItemResults.dat file is a temp file on agent machines that holds the “TimingDetailsStorage” data while a test is running. Once the test run is complete, the data is loaded to the controller, stored in the results repository, and then the temp file is deleted.

This temp file is known to get very large when the timing details setting is on. If you start running out of disk space during a test run, consider cleaning up the disk or turning off timing details.

Memory leak on load test when using HTTPS

Problem: I recently ported some load tests from VS2005 to VS2008 and am noticing a memory leak in VSTestHost.exe when I'm running the load test. After further investigation, it looks like the memory leak seems to occur with requests that have X.509 SSL certificates added to the WebTestRequest.ClientCertificates collection. The load tests were working fine in VS2005.

Resolution: We've analyzed this memory leak and determined that this is a bug in the System.Net.HttpWebRequest class (used to issue Web test requests) that occurs when the Web test target https Web sites. A workaround is to set the Load Test to use the "Connection Pool" connection model.

“Not Trusted” error when starting a load test

When you start a load test, you may get the following error:

“The location of the file or directory xxx is not trusted”

This can occur if you have signed code in your test harness and you make changes to some of the code without resigning it. You can try either one of the below options to attempt to resolve it:

OPTION 1:

1. In the .NET Framework 2.0 Configuration, Go to Runtime Security Policy | Machine | All_Code
2. Right click All_Code, select "New...", and select any name for your new group. Click Next
3. Select URL as your condition
4. Type \\machine_name\shared_folder\assembly.dll or \\machine_name\shared_folder\* and click Next
5. Make sure permission is set to FullTrust
6. Click Next, and Finish
7. Close all your Visual Studio IDEs, restart, and try again

OPTION 2:

```
caspol -machine -addgroup 1 -url file:<location XXX>/* FullTrust -name
FileW
```

Verifying saved results when a test hangs in the “In Progress” state after the test has finished

If you run a test and either the test duration or the number of iterations needed for completion of the test have been reached, but the test stays in the “In Progress” state for a long time, you can check if all of the results have been written to the load test results repository by running this SQL query against the LoadTest database:

```
select LoadTestName, LoadTestRunId, StartTime, EndTime from
LoadTestRun where LoadTestRunId=(select max(LoadTestRunId) from
LoadTestRun);
```

If the EndTime has a non-NULL value then the controller is done writing results to the load test results database and it should be safe to restart the rig (killing anything if needed).

This doesn’t necessarily mean that all results from all agents (if the agents got hung) were successfully written to the load test database, but it does mean that there’s no point in waiting before killing the agents/tests.

Socket errors or "Service Unavailable" errors when running a load test

When running a load test, you might receive several errors similar to:

- `Exception       SocketException             Only one usage of each socket address (protocol/network address/port) is normally permitted`
- `HttpError        503 - ServiceUnavailable             503 - ServiceUnavailable`

These are often due to exhaustion of available connection ports either on the VSTS machine(s) or on the machines under test. To see if this could be happening, open a CMD window on your VSTS machine(s) and on the machine(s) under test, and run the following command:

```
      "netstat –anp tcp"
```

If you see this, then you are suffering from port exhaustion. The following explains what is happening and talks about some ways to deal with it.

TCP establishes connections based on the following items:

- Client port + Client IP = Client Socket
- Server Port + Server IP = Server Socket
- Client Socket + Server Socket = connection

The TIME_WAIT state is a throwback from the old days (well more accurately the default of 4 minutes is the throwback). The idea is that if the client closes a connection, the server puts the socket into a TIME_WAIT state. That way, if the client decides to reconnect, the TCP negotiation does not need to occur again and can save a little bit of time and overhead. The concept was created because creating a TCP connection was a costly operation years ago when networks were very slow).

To get around this issue, you need to make more connections available and/or decrease the amount of time that a connection is kept in TIME_WAIT. In the machine's registry, open the following key and either add or modify the values for the two keys shown:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
]
      "TcpTimedWaitDelay"=dword:0000001e   (30 seconds)
      "MaxUserPort"=dword:0000fffe         (65,535 ports)
```

If you are experiencing the issue on one of the VSTT load machines, you may also need to change the load test connection model to "Connection Pooling" and increase the pool size considerably.

Miscellaneous

Coded web tests and web test plug-ins should not block threads

http://blogs.msdn.com/billbar/archive/2007/06/13/coded-web-tests-and-web-test-plug-ins-should-not-block-the-thread.aspx

Check Your Validation Level in the Load Test Run Settings

By default, all validation rules added to a web test are marked HIGH. By default, all load tests have a validation level of LOW. This means that NONE of the validation rules will run in a load test by default. You either need to lower the level in the web test, or raise the level in the load test.

Add an Analysis Comment

After the load test is complete and you have spent some time analyzing the results, you can add a short one line description and an arbitrarily long analysis comment to be stored permanently with the load test result. To do this, in the load test result viewer, right click and choose the "Analysis" option. This brings up a dialog that allows you to enter your analysis text which is stored in the load test results database when you click OK to close the dialog. NOTE: This can be done while the test is running. You do not need to wait for the test to finish.

Any comments and descriptions added will show up in the "Manage Load Test Results" dialog and will make it much easier to determine which result set maps to the test run you wish to look at.

Programatically Accessing the number of users in Load Tests

In a load test plug-in, you can get the current user load. For an example of this, see Ed Glas's blog post at: http://blogs.msdn.com/edglas/archive/2006/02/06/525614.aspx (listed as "Custom Load Patterns for VSTS" in the offline pages collection). This blog post actually does much more than that, but the line where it updates the current load is:

```
((LoadTestScenario)m_loadTest.Scenarios[0]).CurrentLoad = newLoad;
```

Unfortunately in VSTS 2008 there is still no way to directly access the max load defined for the Scenario from a LoadTestPlugin or anywhere else. You could make this a LoadTest Context parameter (but you'd have to make sure that the value of the Load Test context parameter stays in sync with the max user load specified in the Scenario.

In VSTS 2008 SP1, we are adding the ability for a load test plugin to access or update the properties of a Load Test Scenario's load pattern. A description of that enhancement with examples is contained in the above link.

Load Test Rig Consideration

Known Issues

Multi-proc boxes used as agents should have .NET garbage collection set to server mode

To enable your application to use Server GC, you need to modify either the **VSTestHost.exe.config** or the **QTAgent.exe.config**. If you are not using a Controller and Agent setup, then you need to modify the **VSTesthost.exe.config**. If you are using a controller and agent, then modify the **QTAgent.exe.config** for each agent machine. Open the correct file. The locations are

```
VSTestHost.exe.config - C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE
QTAgent.exe.config - C:\Program Files\Microsoft Visual Studio 9.0 Team Test Load
Agent\LoadTest
```

To enable gcserver you need to add the following highlighted line in the runtime section:

```
<?xml version ="1.0"?>
<configuration>
    <runtime>
        <gcServer enabled="true" />
        <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
            <probing
privatePath="PrivateAssemblies;PublicAssemblies"/>
        </assemblyBinding>
    </runtime>
</configuration>
```

Slowness Restarting a Test Rig with Agents Marked as “Offline”

If you have agent machines that are either disabled (powered off, service stopped, etc) or that no longer exist, but you only mark them as “Offline” in the “Administer Test Controllers” dialog, restarting the rig will take a long time. The controller will attempt to contact all agents listed in the dialog regardless of their status, and it will take approximately one minute or more for each missing machine.

VSTS does not appear to be using more than one processor

If you are running a load test on a multi processor machine but notice that only one processor is being used, this is due to the fact that you are running the test as “<Local – no controller>”. VSTS will only use multiple processors on an Agent/Controller setup. This is by design due to licensing considerations. In order to take advantage of multi-proc systems, please use an agent and controller setup. It is possible to setup the controller and agent on the same machine as VSTS.

Multiple IP addresses can cause tests in a rig to not start

Problem: When we run the tests from this controller the tests just start with pending state and nothing else happens. The IP Address of the controller and agents have been changed from the initial setup.

Local Machine Resolution: I think the problem may be that you effectively have two IP addresses on this machine. The following entries in the controller log confirm my suspicion that this is the problem:

```
[I, 2972, 11, 2008/06/26 13:02:59.780] QTController.exe: ControllerExecution: Calling
back to client for deployment settings.

[E, 2972, 11, 2008/06/26 13:06:51.155] QTController.exe: StateMachine(RunState):
Exception while handling state Deploying: System.Net.Sockets.SocketException: A
connection attempt failed because the connected party did not properly respond after a
period of time, or established connection failed because connected host has failed to
respond 65.52.230.25:15533
```

This is exactly the type of error message we see when the controller communication with Visual Studio fails because the client has two IP addresses: To configure your Visual Studio installation to communicate with the controller, try this:

In regedit:

- Find the key:
  HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\VisualStudio\9.0\EnterpriseTools\QualityTools
- Add a new key under the above key named “ListenPortRange”
- In the key “ListenPortRange”, add a new string value with the name “BindTo” and the IPv4 address for the client (65.52.230.25 in your case) as the BindTo value.

Test Rig Resolution:

Read the following support article for the steps to resolve this issue on a test rig: http://support.microsoft.com/kb/944496

IP Address Spoofing Information

IP Address Spoofing anatomy (how it works)

VSTT currently limits the number of unique IP addresses to 256 per agent. In most testing situations, this will be plenty of addresses. The main place where this limitation might impact you is if you are running a large test where every single user must have a separate IP Address for some sort of session state.

If your load test uses "Connection per User" (which is the default) in VSTS 2008 and you set the size of the IP address range equal to the number of virtual users per agent, then you will get a unique IP address for each user, however it will not stick across iterations for each user. You must use "Connection Pool Model" in order for your unique IP Address to persist across iterations.

The biggest problem with assigning unique IP Addresses to every user is that currently the IP switching configuration limits you to a range of 256 IP addresses per agent, which would mean you would also be limited to 256 virtual users per agent. One solution is to use VMs to get multiple load test agents on a single physical machine.

IP Addresses used for spoofing are not permanent

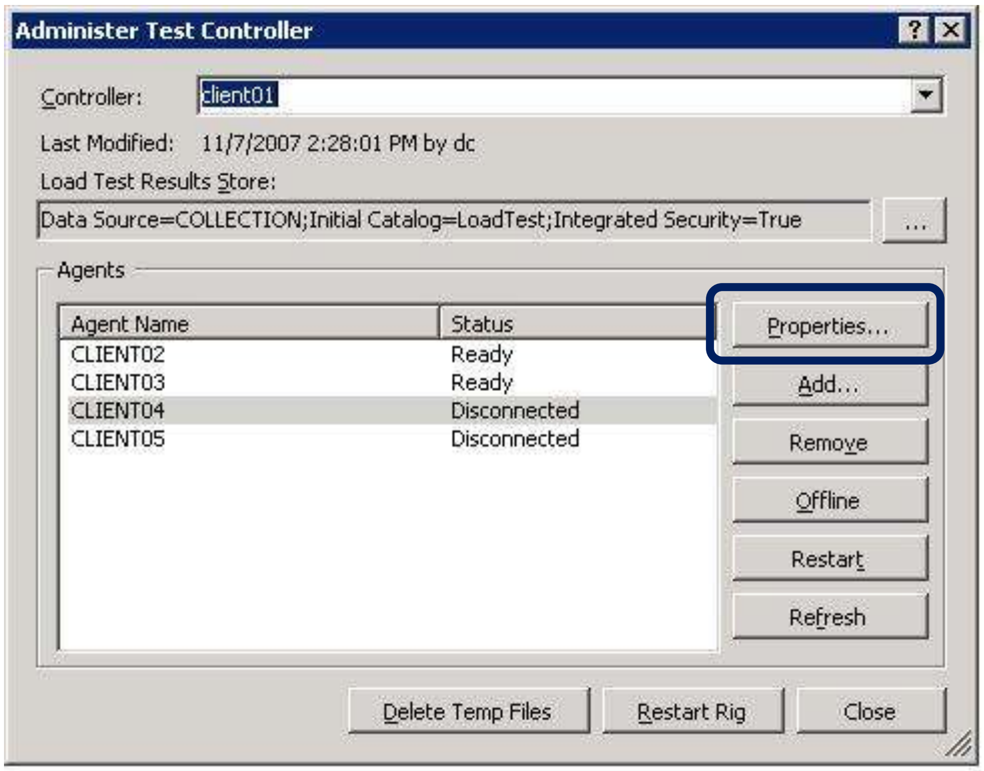
When you choose to use multiple IP addresses from each agent machine during load testing (known as IP address spoofing), most testing tools require you to add those IP addresses to the NIC of the machine, and they are always available and always show up on the machines. VSTS allows you to set a range of IP addresses directly in the test project. Then VSTS dynamically adds the addresses to the agent(s) when the test run starts, and removes them when the test run stops. . If you need to perform IP spoofing, a controller/agent setup is required.

How to Setup IP Spoofing

There are 2 parts to setting up IP Spoofing. First, you must configure the Test Rig Agents to use IP Spoofing. Then you must tell the Load Test itself that it should take advantage of that. Here are the steps and the pitfalls involved:

Setting up the agents

1. Open up the Test Rig Administration dialog (Test -> Administer Test Controller)
2. Highlight each of the agents and bring up the Properties for the agent
3. Fill out all of the appropriate information (as outlined in the picture below)



Controller: client01

Last Modified: 11/7/2007 2:28:01 PM by dc

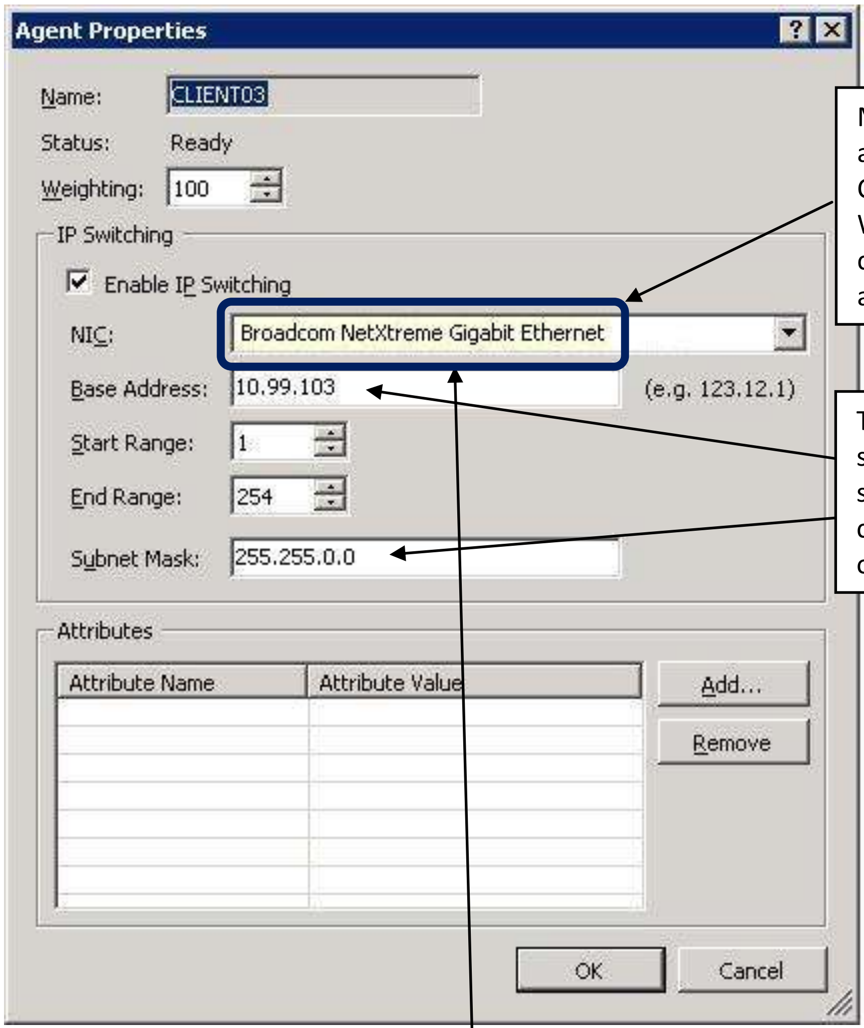
Load Test Results Store:

Data Source=COLLECTION;Initial Catalog=LoadTest;Integrated Security=True

Agents

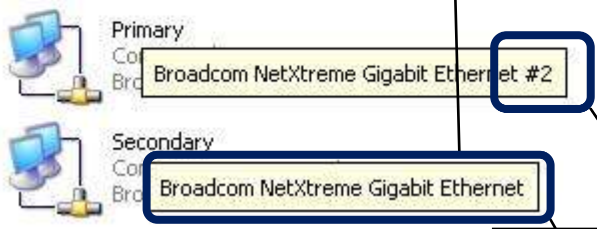
Agent Name	Status
CLIENT02	Ready
CLIENT03	Ready
CLIENT04	Disconnected
CLIENT05	Disconnected

Where to configure Agent Properties



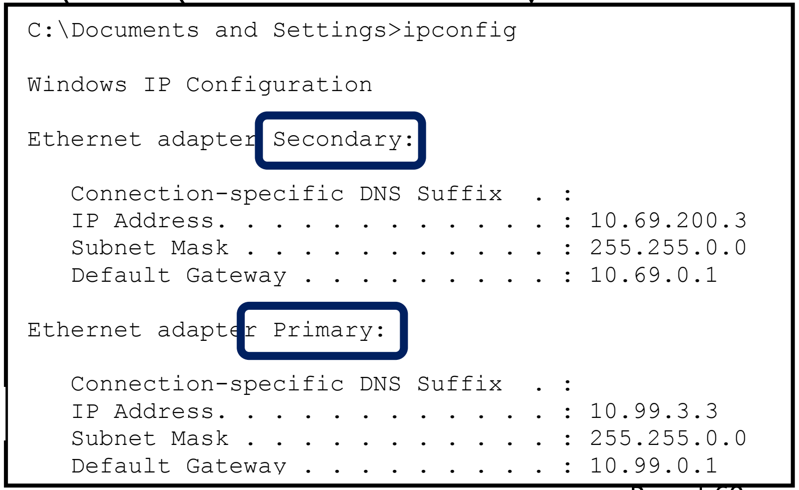
Make sure you pick the correct adapter here. Use the Network Connections properties built into Windows along with the IPCONFIG command to see which NIC is assigned to what subnet (see below).

The base address is 3 octets and should be representative of the subnet you are on. If you are using a class B subnet, you still need a third octet for the base.



Primary
Broadcom NetXtreme Gigabit Ethernet #2
Secondary
Broadcom NetXtreme Gigabit Ethernet

The output from the IPCONFIG command in a CMD window.



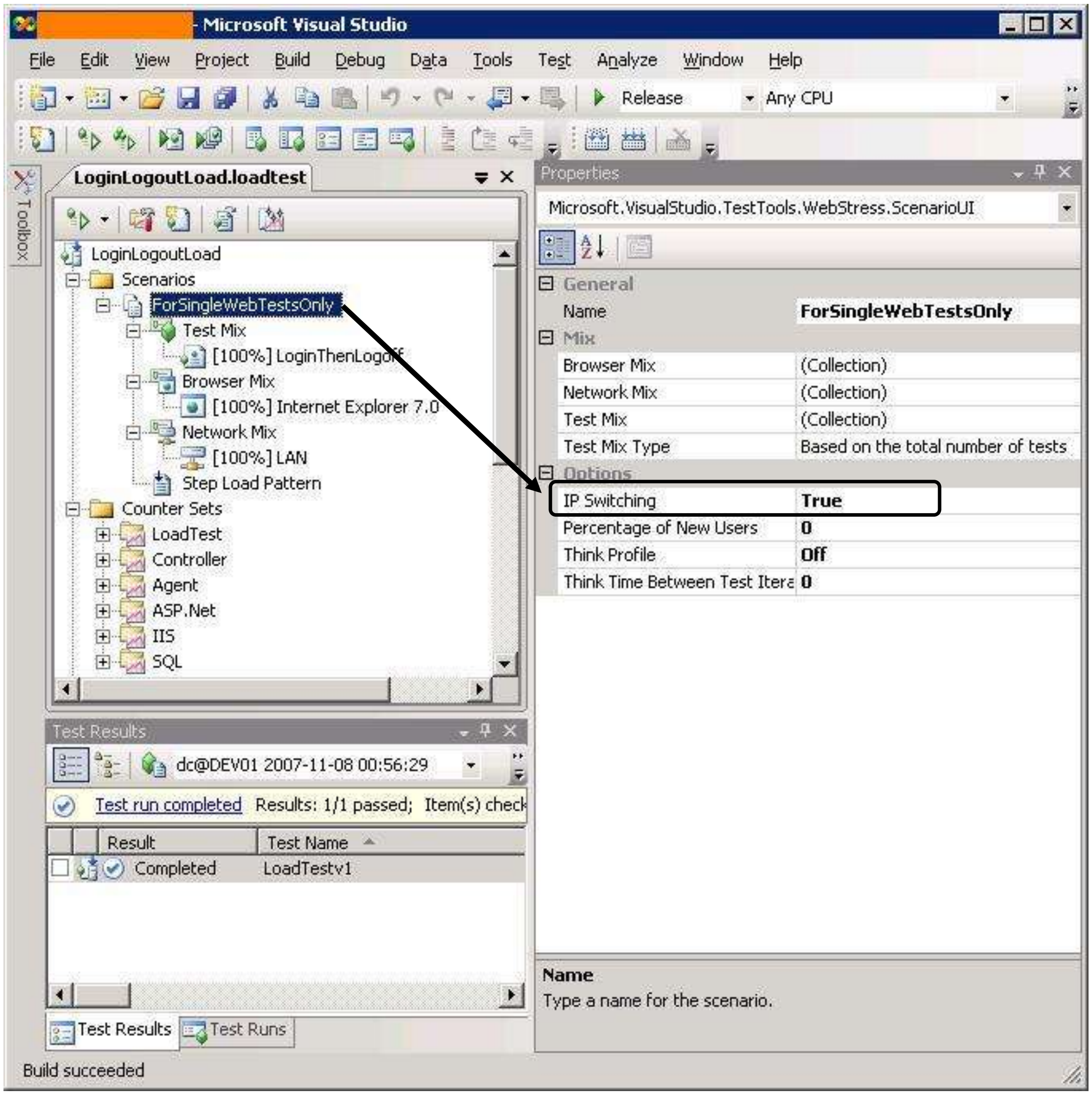
```
C:\Documents and Settings>ipconfig

Windows IP Configuration

Ethernet adapter Secondary:

   Connection-specific DNS Suffix  . :
   IP Address. . . . . . . . . . . . : 10.69.200.3
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . : 10.69.0.1

Ethernet adapter Primary:

   Connection-specific DNS Suffix  . :
   IP Address. . . . . . . . . . . . : 10.99.3.3
   Subnet Mask . . . . . . . . . . . : 255.255.0.0
   Default Gateway . . . . . . . . . : 10.99.0.1
```

The information as shown in the Network Connections dialog box in Windows. You may need to hover the mouse over the NIC to see the entire

Getting the proper IP Address info for spoofing

Setting up The Load Test

Once the test rig is setup, you can configure which Load Test will actually use IP Spoofing by setting the correct property for the Load Test:



Microsoft.VisualStudio.TestTools.WebStress.ScenarioUI

General	
Name	ForSingleWebTestsOnly
Mix	
Browser Mix	(Collection)
Network Mix	(Collection)
Test Mix	(Collection)
Test Mix Type	Based on the total number of tests
Options	
IP Switching	True
Percentage of New Users	0
Think Profile	Off
Think Time Between Test Itera	0

Name
Type a name for the scenario.

dc@DEV01 2007-11-08 00:56:29

Test run completed Results: 1/1 passed; Item(s) check

Result	Test Name
Completed	LoadTestv1

Where to enable IP Switching for the Load Test Itself (after configuring the agents to use it)

Troubleshooting the VSTS Load Testing IP Switching Feature

1) Make sure that the Agent Service is running as a user than is an admin on the agent machine; this is required because the agent service attempts to configure the IP addresses specified in the agent properties on the chosen NIC, and admin permission is required to do this.

2) Make sure that none of the IP addresses in the range specified for a particular agent are already configured on the chosen NIC.

3) Enable verbose logging for the Agent Service:

* Edit the file QTAgentService.exe.config: (located at: <Program Files>\Microsoft Visual Studio 9.0 Team Test Load Agent\LoadTest\QTAgentService.exe.config)

* Change:
  <add key="CreateTraceListener" value="no"/> to "yes"
* Change:
  <add name="EqtTraceLevel" value="3" /> to "4"
* Restart the Load Test Agent service

* The file "VSTTAgent.log" will be created in the same directory as QTAgentService.exe.config.

* Re-run the load test with verbose logging configured, and look for lines in the log file that contain the text: *"Attempting to configure IP address:"* and *"Configured IP address:"* This will tell you whether or not you the agent service is attempting to configure the IP address you've specified. If you see the "Configured IP address:" line, it has succeeded in configuring this IP address. If not, there should be some error logged.

If you have verified the items in step 1 & 2 above, and the log indicates that the configuration of the IP address is failing but you cannot determine the cause of the failure from the error message in the log (or if there is no error message in the log), post a new thread to the Web and Load testing forum, or open a Microsoft Support incident for further assistance, and provide details on the setup including the relevant portions of the log file.

4) Make sure that the load test you are running is set to use IP Switching: Click on each of the "Scenario" nodes in the load test editor, go to the property sheet, and verify that the "IP Switching" property is set to True (normally it should be since this is the default, but it's worth checking).

5) Enable verbose logging for the Agent process.

If the log file created in step 3 shows that the IP addresses are being successfully configured, the next step is to check the agent process log file to verify that the load test is actually sending requests using those IP addresses.

To enable verbose logging for the agent process:
* Edit the file QTAgent.exe.config: (located at <Program Files>\Microsoft Visual Studio 9.0 Team Test Load Agent\LoadTest\QTAgent.exe.config)

* Change:
    <add key="CreateTraceListener" value="no"/> to “yes”
* Change:
    <add name="EqtTraceLevel" value="3" /> to “4”

* The file "VSTTAgentProcess.log" will be created in the same directory as QTAgent.exe.config.
* Re-run the load test, and look for lines in the log file that look something like: "*Bound request on connection group M to IP address NNN.NNN.NNN.NNN*" If verbose logging is enabled and these lines are present in the log file, IP Switching should be working.

6) If the number of unique IP addresses being used as shown by the log entries in step 5 is less than the number in the range that was configured, it could be because your load test is configured to use a connection pool with a smaller number of connections than the number of IP addresses specified. If this is the case, you can increase the size of the connection pool, or switch to "Connection per User" mode in the load test's run settings properties.

Miscellaneous and “How It Works”

(BUG) Changes made to Web Test Plugins may not show up properly

If you have a plugin that is part of the same project as a declarative web test, and you make changes in the plugin, you may not always see those changes reflected in the test run. For instance, if you have a plugin that writes a certain string out to an event log, and you change the string in the plugin, you still see the old string value in the event log. This is a known issue and may be fixed in VSTT 2008 SP1 (it is not in the beta release of SP1). In order for the bug to appear, the following conditions must be met:

- You must be running on a controller/Agent test rig
- Your web test must be declarative (bug does not occur with coded web tests)
- You must have a “Test Results” folder in the root of your solution folder

If you are experiencing the bug, you can work around it by:

- Generating a coded web test
- Renaming or deleting the “Test Results” folder
- Changing the test project’s location for the “Test Results” folder

Startup of tests on a Rig with Agents on a Slow Link

The load test does not actually start on any agents until deployment of all files has occurred to all agents (by the way, this means that the slow up start of a load test on a rig with many agents could have been caused by slow deployment to one or more agents).

Performance Counter Considerations on Rigs with slow links

Having a slow WAN between the controller and agents may definitely cause some timeouts or delays in performance counter collection.   Each performance counter category is read in a separate operation: that's one method call at the level of the .NET classes that we call, and I don't know if each call results in just one or more than one network read.
There are some timeout settings for performance counter collection that you can change by editing the QTController.exe.config file and adding these lines:

```
<appSettings>
   <add key="LoadTestCounterCategoryReadTimeout" value="9000"/>
   <add key="LoadTestCounterCategoryExistsTimeout" value="30000"/>
</appSettings>
```

The values are in ms, so 9000 is 9 seconds.   If you make this change, also change the load test sample rate to be larger than this: at least 10 or preferably 15 seconds, and yes with many agents located far from the controller, it is recommended to delete most of the categories in the Agent counter set (perhaps just leave Processor and Memory).

The .NET API that used to read the performance counters is PerformanceCounterCategory.ReadCategory(), so the entire category is read even if the counter set definition only includes one counter and one instance.   This is a limitation at the OS level in the way performance counters are read.

The defaults in VSTS 2008 are:

- LoadTestCounterCategoryReadTimeout: 2000 ms (2 seconds)
- LoadTestCounterCategoryExistsTimeout: 10000 ms

Performance Data Collection and usage

Setup and Configuration

Increase the performance counter sampling interval for longer tests

Choose an appropriate value for the “Sample Rate” property in the Load Test Run Settings based on the length of your load test. A smaller sample rate, such as the default value of five seconds, requires more space in the load test results database. For longer load tests, increasing the sample rate reduces the amount of data collected.

Here are some guidelines for sample rates:

Load Test Duration	Recommended Sample Rate
< 1 Hour	5 seconds
1 - 8 Hours	15 seconds
8 - 24 Hours	30 seconds
> 24 Hours	60 seconds

Consider including Timing Details to collect percentile data

There is a property on the Run Settings in the Load Test Editor named "Timing Details Storage". If Timing Details Storage is enabled, then the time to execute each individual test, transaction, and page during the load test will be stored in the load test results repository. This allows 90th and 95th percentile data to be shown in the load test analyzer in the Tests, Transactions, and Pages tables. The amount of space required in the load test results repository to store the Timing Details data may be very large, especially for longer running load tests. Also, the time to store this data in the load test results repository at the end of the load test is longer because this data is stored on the load test agents until the load test has finished executing at which time the data is stored into the repository. For these reasons, Timing Details is disabled by default. However if sufficient disk space is available in the load test results repository, you may wish to enable Timing Details to get the percentile data. Note that there are two choices for enabling Timing Details in the Run Settings properties named "StatisticsOnly" and "AllIndividualDetails". With either option, all of the individual tests, pages, and transactions are timed, and percentile data is calculated from the individual timing data. The difference is that with the StatisticsOnly option, once the percentile data has been calculated, the individual timing data is deleted from the repository. This reduces the amount of space required in the repository when using Timing Details. However, advanced users may want to process the timing detail data in other way using SQL tools, in which case the AllIndividualDetails option should be used so that the timing detail data is available for that processing.

Customizing the Available Microsoft System Monitor counter sets

The counter set templates for VSTS are located in the following directory (assuming a typical install):

```
C:\Program Files\Microsoft Visual Studio
9.0\Common7\IDE\Templates\LoadTest\CounterSets
```

These files are standard XML files and can be modified to allow for quick and easy re-use of custom sets. It is recommended that you copy the counter set you wish to enhance and add the name CUSTOM to it so you will always remember that it is a custom counter set. Or you can create your own totally independent counter set. The following shows the layout of the file:

```
<?xml version="1.0" encoding="utf-8"?>
    <CounterSet Name="Custom" CounterSetType="Custom Set">
      <CounterCategories>
        <CounterCategory Name="Memory">
          <Counters>
            <Counter Name="% Committed Bytes In Use" />
            <Counter Name="Available Mbytes" />
          </Counters>
        </CounterCategory>
        <CounterCategory Name="Processor">
          <Counters>
            <Counter Name="% Processor Time">
              <ThresholdRules>
                <ThresholdRule
                    Classname="Microsoft.VisualStudio.TestTools.WebStress.Rules.Thresh
                    oldRuleCompareConstant,
                    Microsoft.VisualStudio.QualityTools.LoadTest">
                  <RuleParameters>
                    <RuleParameter Name="AlertIfOver" Value="True" />
                    <RuleParameter Name="WarningThreshold" Value="80" />
                    <RuleParameter Name="CriticalThreshold" Value="95" />
                  </RuleParameters>
                </ThresholdRule>
              </ThresholdRules>
            </Counter>
          </Counters>
          <Instances>
            <Instance Name="*" />
          </Instances>
        </CounterCategory>
      </CounterCategories>
    </CounterSet>
```

This all needs to be on one line. You can copy all of the code here and paste it into notepad and it should be formatted properly.

Changing the default counters shown in the graphs during testing

If you want to change the default set of counters that show up in the graphs when you start a test, you can go into each of the .counterset XML files (same directory as above) and set or add to the **DefaultCounter** entries in the following section (at the bottom of the files):

```
<DefaultCountersForAutomaticGraphs>
  <DefaultCounter CategoryName="Memory" CounterName="Available MBytes"/>
</DefaultCountersForAutomaticGraphs>
```

Consider enabling SQL Tracing Throught the Load Test Instead of Separately

There is a set of properties on the Run Settings in the Load Test Editor that allow the SQL tracing feature of Microsoft SQL Server to be enabled for the duration of the load test. If enabled, this allows SQL trace data to be displayed in the load test analyzer on the "SQL Trace" table available in the Tables dropdown. This is a fairly easy-to-use alternative to starting a separate SQL Profiler session while the load test is running to diagnose SQL performance problems. To enable this feature, the user running the load test (or the controller user in the case of a load test run on a rig) must have the SQL privileges needed to perform SQL tracing, and a directory (usually a share) where the trace file will be written must be specified. At the completion of the load test, the trace file data is imported into the load test repository and associated with the load test that was run so that it can be viewed at any later time using the load test analyzer.

Collecting SQL counters from a non-default SQL instance

If you want to collect performance counters from a SQL Server instance while running a load test, you can do this easily by selecting checking the SQL counter set in the "Manager Counter Sets" dialog in the VSTS load test editor. Doing this includes the default counter set for SQL Server in your load test. The performance counter category names that are specified in this counter set begin with "SQLServer:": for example "SQLServer:Locks". However, if you are trying to monitor a SQL Server instance that is not the default SQL server instance, the names of the performance counter categories for that instance will have different category names. For example, if your SQL server instance is named "INST_A", then this performance counter category will be named "MSSQL$INST_A:Locks". To change the load test to collect these performance counters, the easiest thing to do is open the .loadtest file with the XML editor or a text editor and replace all instances of "SQLServer:" by "MSSQL$INST_A:Locks" (correcting the replacement string for your instance name).

Known Issues

Error “Failed to load results from the load test results store”

“Unable to cast object of type ‘System.DBNull’ to type ‘System.Byte[]’” error when trying to retrieve load test results from the DB inside VSTS.

This error will occur if you get a NULL value in the LoadTest column of the LoadTestRun table. To fix it, go to the table and delete the row that has the NULL value. The occurrence of this issue should be extremely rare.

Miscellaneous and “How It Works”

How and where Performance data gets collected

There are two types of data collected by VSTS during a test run: Real perfmon counters and pseudo perfmon counters. All real perfmon counters are collected directly by the VSTS Controller machine.

In the Load Test editor, all of the performance counter categories that start with “LoadTest:” (see the LoadTest counter set in the load test editor) is data that is collected on the agents by the load test runtime engine. These are not real Perfmon counters in the sense that if you try to look at them with Perfmon you won’t see them, though we make them look like Perfmon counters for consistency in the load test results database and display. The agents send this some of this data (see below) in messages to the controller every 5 seconds which rolls up the agent (e.g. Requests / sec across the entire rig rather than per agent). The controller returns the rolled up results to Visual Studio for display during the run and also stores them in the load test results database.

[Requests Per Second Counters] The VSTT RPS does not count cached requests, even though VSTS is sending an http GET with if-modified-since headers.

What data is sent every 5 seconds? we do everything possible to limit how much data is sent back in that message. What we do send back is the average, min, max values for all of the pseudo performance counters in the categories that start with “LoadTest:” that you see under the “Overall”, “Scenarios” and “Errors” nodes in the load test analyzer tree (nothing under the “Machines” node). Note that the biggest factor in the size of these result messages is the number of performance counter instances, which for Web tests is mostly determined by the number of unique URLs reported on during the load test. We also send back errors in these 5 seconds messages, but details about the failed requests are not sent until the end of the test, so tests with lots of errors will have bigger messages. Lastly, we only send back metadata such as the category names and counter names once and use numeric identifiers in subsequent messages, so the messages at the start of the load test may be slightly larger than later messages.

One thing you could do to reduce the size of the messages is to reduce the level of reporting on dependent requests. You could do this by setting the “RecordResult” property of the WebTestRequest object to false. This eliminate the page and request level reporting for that request, but you could add a transaction around that request single request and that would really match the page time for that request

How 90% and 95% response times are calculated

Within the load test results summary page, the percentile values mean that:

- 90% of the total transactions were completed in **less than** <*time*> seconds
- 95% of the total transactions were completed in **less than** <*time*> seconds

The calculation of the percentile data for transactions is based not on the sampled data that is shown in the graph, but on the individual timing details data that is stored in the table LoadTestTransactionDetail. The calculation is done using a SQL stored procedure that orders the data by the slowest transaction times, uses the SQL "top 10 percent" clause, to find the 10% of the slowest transactions then uses the min() function on that set of rows to get the for the value for the 90th percentile value for example. The stored procedure in the LoadTest database that does this is "`Prc_UpdateTransactionPercentiles`".

Transaction Avg. Response Time vs. Request Avg. Response Time

for each HTTP request (including each dependent request) there is a request response time, and these are all averaged to get the "Avg. Response Time" that appears on the default graph and on the Requests table in the load test analyzer. There is also the "Avg. Page Time" (seen on the Pages table and can be graphed, but is not be default) that is the average time to download the request that is in the web test plus the time to download all dependents (dependents may be downloaded in parallel). Then for transactions, there are two counters: "Avg. Response Time" and "Avg. Transaction Time". The former is the average of the sum of all of the page times (without the think times), and the latter is the same but includes the think times.

For more descriptions see this online doc page: http://msdn.microsoft.com/en-us/library/ms404656.aspx.

Application Profiling

Using the VSTS Application Profiler

http://www.codeguru.com/cpp/v-s/devstudio_macros/visualstudionet/article.php/c14823__1/

VSTS 2008 Application Profiler New Features

The VSTS perf team has added some blog posts outlining new features of the VSTS profiler and how to use them. These features include a quick tool to find “hotspots” in your app, and the ability to use performance counters to enhance your profiler diagnosis. See the following links to get this info:

http://blogs.msdn.com/profiler/archive/2007/10/19/articles-on-new-visual-studio-team-system-2008-profiler-features.aspx

Load Test Results Store Information

Considerations for the location of the Load Test Results Store

When the Visual Studio Team Test Controller is installed, the Load Test Results Store is set up to use an instance of SQL Express that is installed on the controller computer. SQL Express is limited to using a maximum of 4 GB of disk space. If you are going to run many load tests and want to keep the results for a while, you should consider configuring the Load Test Results Store to use an instance of the full SQL Server product if available. See the Visual Studio Team Test documentation for instructions on setting up the database to be used as the Load Test Results Store.

How to clean up results data from runs that did not complete

If you have a Load Test Run that abnormally aborts and does not show data in a consistent manner (or does not show up in the list of runs as either completed or aborted), you can use the following query on the SQL repository server to clean up the database:

```
update LoadTestRun set Outcome='Aborted' where Outcome='InProgress'
```

The Outcome field is left blank until the test either completes or is manually aborted. Any test results in the DB cannot be accessed through the GUI until the Outcome field has one of the two values 'Completed' or 'Aborted'

InstanceName field in results database are appended with (002), (003), etc.

Question: In the LoadTest databases, the Instance Names are sometimes appended with “(002)”, etc. For example, I have a transaction called “Filter Render Request” and in the load test database I have two transactions. Also, I have a URL pointing to RenderWebPartContent and I have several entries. Can someone give me a quick explanation ?

Answer: To make a long story short it is a unique identifier that is used mostly internally to distinguish between cases where you have the same test name in two different scenarios in the load test or the same page name (simple file name) in different folders in two different requests.

Layout for VSTS Load Test Results Store

http://blogs.msdn.com/billbar/articles/529874.aspx

How to view Test Results from the GUI

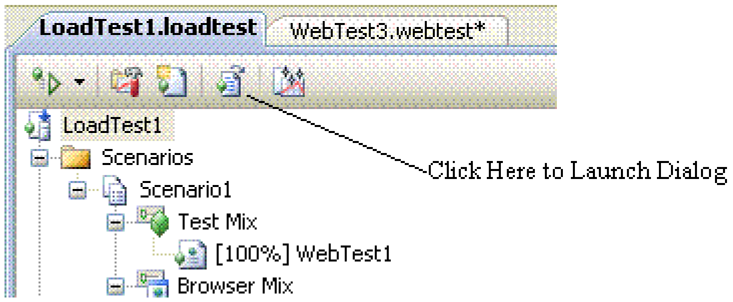
http://blogs.msdn.com/slumley/pages/managing-load-test-results.aspx

SQL Server Reporting Services Reports available for download

http://blogs.msdn.com/slumley/pages/load-test-reports.aspx

How to move results data to another system

VSTS 2008 introduces a GUI results manager. The manager works on the Load Test Results Store that is currently specified in the “Administer Test Controller” dialog box, or on the local repository. To open the results manager, you must have a load test opened and set as the active window. Then click on the icon shown below:



How to launch the “Manage Load Test Results” dialog box

Once in the manager, you choose a controller name from the drop down list (or <local> if you want the results from the local database) and the manager will populate with the tests it finds. You can select whatever test results you wish to move, and then choose “export” to move them into a file (compressed with an extension of .ltrar). That file can be moved to another machine and then imported into a new results store.

Test Customization

Creating Validation Rules, Extraction Rules and Plugins

The following link will give information on creating plugins, custom validation rules, custom extraction rules, etc.

http://msdn2.microsoft.com/en-us/library/ms182553.aspx (NOTE: This link is to a series of items and therefore does not have an associated offline PDF file).

Manually moving the data cursor

Add the following line of code to force the parameter database to advance by one row. This is useful if you need to loop through sections of code in a single iteration and want to use different data.

```
this.MoveDataTableCursor(<"DataSource1">, <"Products">);
```

Items changed or fixed in VSTS 2008 SP1

Content-Length header not available in Web Request Object

Currently the web request header "Content-Length" in not in the WebTestRequest object. This is expected to be changed in SP1

SharePoint file upload test may post the file twice

If you have a web test that posts a file to a SharePoint site, the test may try to post the file twice. SharePoint will only process one copy, but the request time and upload size will be incorrect due to the double attempt. This occurs if you are using integrated authentication. The client requests a POST expecting a 100-continue response. It gets a 404 instead (this is expected behavior). However, instead of VSTS restarting the request with credentials, it continues posting the initial request (which SharePoint ignores). When the initial request is done, VSTS will re-post with credentials, and this post will succeed. A fix is available in SP1.

Some Hidden Fields are not parameterized within AJAX calls

When recording web tests with AJAX panel updates, you may find some FORM POST parameters where HIDDEN values (such as VIEWSTATE) are not parameterized. From an email thread:

The problem is that in the Microsoft-Ajax partial rendering (update panel) responses, hidden fields can appear in two places: a field that is marked by the type "|hiddenField|" (where we were looking), but also in a regular hidden field input tag in the HTML within an "|updatePanel|" field in the Ajax response (which we were not looking at).

A fix is being worked on and may appear in SP1. In the meantime, to work around the issue, simply remove the hard-coded value and replace it with a parameterized value: **{{$HIDDEN0.__VIEWSTATE}}** (where the bucket (0,1,2, etc matches the bucket of the other HIDDEN parameters in the request)

(FIX) Unit Test threading models and changing them

The default threading model for unit tests is STA. The fix in SP1 was to have load tests honor this setting (unit test in a load test would not honor the ApartmentState property). See the following blog for more info:

http://blogs.msdn.com/irenak/archive/2008/02/22/sysk-365-how-to-get-your-unit-tests-test-project-in-visual-studio-2008-a-k-a-mstest

Bug in VSTS 2008 SP1 causes think time for redirected requests to be ignored in a load test

When a web test is run in a load test, any test requests that result in redirects suffer from a timing bug. Any think time that is specified on the request is ignored. This is fixed in a POST-SP1 hotfix:

KB 956397 (http://support.microsoft.com/kb/956397/en-us)

http://blogs.msdn.com/billbar/archive/2008/08/04/bug-in-vsts-2008-sp1-causes-think-time-for-redirected-requests-to-be-ignored-in-a-load-test.aspx

New Load Test Plugin Enhancements in VSTS 2008 SP1

http://blogs.msdn.com/billbar/pages/load-test-api-enhancements-in-vsts-2008-sp1-beta.aspx

Four New Methods added to the WebTestPlugin Class for 2008 SP1

http://blogs.msdn.com/billbar/pages/web-test-api-enhancements-available-in-vsts-2008-sp1-beta.aspx

General Commands and tricks (not VSTS specific)

How to add spaces in alias names for logparser

Use square brackets around an alias name to allow spaces in the names. Example: [User Data]

Logparser WEB Queries

Count (and percentage) of status codes from Web Logs

```
-i:IISW3C -recurse:-1 -Q:on "SELECT sc-status, COUNT(*), MUL(PROPCOUNT(*),100.0) AS
Percentage INTO StatusCount.txt FROM ex*.log GROUP BY sc-status ORDER BY sc-status”
```

Breakdown of web status codes by pagetype from Web Logs

```
-i:IISW3C -recurse:-1 -Q:on "SELECT EXTRACT_EXTENSION(TO_UPPERCASE(cs-uri-stem)) AS
PageType, sc-status, COUNT(*) AS Amount INTO StatusCodes.txt FROM ex*.log GROUP BY sc-
status, PageType ORDER BY sc-status ASC" -o:TSV
```

Number of hits by pagetype

```
-i:IISW3C -recurse:-1 -Q:on "SELECT EXTRACT_EXTENSION(TO_UPPERCASE(cs-uri-stem)) AS
PageType, COUNT(*) AS Amount INTO pagetype.txt FROM ex*.log GROUP BY PageType ORDER BY
Amount DESC" -o:TSV
```

List of requests asking for non existant pages

```
-i:IISW3C -recurse:-1 -Q:on "SELECT DISTINCT cs-uri-stem AS Url USING sc-status AS
statuscode INTO not-found.txt FROM ex*.log WHERE statuscode = 404" -o:TSV
```

Top 10 slowest page responses

```
-i:IISW3C -recurse:-1 -Q:on "SELECT TOP 10 MAX(time-taken) AS Processing-Time,
AVG(time-taken) AS Average, MIN(time-taken) AS Minimum, cs-uri-stem AS Url, COUNT(cs-
uri-stem) AS PageCount INTO longrunning.txt FROM ex*.log GROUP BY cs-uri-stem ORDER BY
Average DESC" -o:TSV
```

Average Max and Min time taken for each page type

```
-i:IISW3C -recurse:-1 -Q:on "SELECT EXTRACT_EXTENSION(cs-uri-stem) as Type, AVG(time-
taken) AS Average, MAX(time-taken) AS Maximum, MIN(time-taken) AS Minimum INTO
PageTimes.txt FROM ex*.log WHERE time-taken &amp;amp;gt; 0 GROUP BY Type ORDER BY
Average DESC"
```

Requests and Total Bytes per hour

```
-i:IISW3C -recurse:-1 -Q:on "SELECT QUANTIZE(TO_TIMESTAMP(date, time), 3600) AS Hour,
COUNT(*) AS Total, SUM(sc-bytes) AS TotBytesSent INTO HitsByHour.txt FROM ex*.log
GROUP BY Hour ORDER BY Hour" -o:TSV
```

Get ASPX Pages

```
-i:IISW3C -recurse:-1 -Q:on "SELECT MAX(time-taken) AS MaxTime, AVG(time-taken) AS
Average, MIN(time-taken) AS Minimum, COUNT(cs-uri-stem) AS PgCount,
MUL(PROPCOUNT(*),100.0) AS Percent, TO_UPPERCASE(cs-uri-stem) AS Url INTO
ASPX_Pages.txt FROM ex*.log WHERE EXTRACT_EXTENSION(TO_UPPERCASE(cs-uri-stem)) =
'ASPX'  GROUP BY Url ORDER BY PgCount DESC" -o:TSV
```

List and count of pages returning a status code of 500

```
-i:IISW3C -recurse:-1 -Q:on "SELECT cs-uri-stem, sc-status, COUNT(*) FROM ex*.log
WHERE sc-status=500 GROUP BY cs-uri-stem, sc-status ORDER BY cs-uri-stem" -o:TSV
```

LogParser Non-Web Queries

Parsing Event Viewer files on Vista with LogParser

You need to convert the files to Vista format. The following command line will do this:

```
wevtutil epl app.evtx app.evt /lf:true
```

Query For Text Strings in a file

```
-i:TEXTLINE "SELECT LTRIM(extract_token(text, 1,'Text to find')) as string FROM *.txt
WHERE string is not null"
```

Pulling data from inside the body string of event viewer logs

```
logparser -i:evt "SELECT extract_prefix(extract_suffix(Strings,0,'left text'),0,'right
text') as String INTO optimizer.txt FROM *.EVT WHERE Strings LIKE '%Optimizer
Results%'" -q:ON
```

(variation) Pulling data from inside the body string of event viewer logs constrained by timeframe

```
logparser -i:evt -q:ON "SELECT Count(*) AS Qty, SUBSTR(extract_suffix(Message, 0,
'Message :'), 0, 75) as String FROM Error! Hyperlink reference not
valid.name>\Application WHERE SourceName LIKE '%Enterprise%' AND Message LIKE
'%Timestamp: %' AND TimeGenerated > TIMESTAMP ('2008-06-06 07:23:15', 'yyyy-MM-dd
hh:mm:ss' ) GROUP BY String ORDER BY Qty DESC"
```

List of exceptions from saved event logs searching for keywords in the text output

```
-I:evt "SELECT QUANTIZE(TimeGenerated, 3600) AS Hour, COUNT(*) As Total, ComputerName
FROM *.evt WHERE EventID = 100 AND strings like '%overflow%' GROUP BY ComputerName,
hour"
```

Logparser command for querying netstat

```
netstat.exe -anp TCP | LogParser "SELECT [Local Address] AS Server,[Foreign
Address] AS Client,State FROM STDIN WHERE Server LIKE '%:443' OR Server LIKE
'%:80'" -i:TSV -iSeparator:space -nSep:2 -fixedSep:OFF -nSkipLines:3 -o:TSV -
headers:ON
```

Command to query Active Directory®

```
Logparser -i:ADS "SELECT * FROM 'LDAP://Redmond/CN=Microsoft.com FTE,OU=Distribution
Lists,DC=redmond,DC=corp,DC=microsoft,DC=com'" -objClass:user
```

Command to query IIS and get site configuration information

```
Logparser "select * from IIS://localhost"
```

Command to query Netmon file and list out data on each TCP conversation

```
LogParser -fMode:TCPConn -rtp:-1 "SELECT DateTime, TO_INT(TimeTaken) AS Time,
DstPayloadBytes, SUBSTR(DstPayload, 0, 128) AS Start_Of_Payload INTO IE-Take2.txt FROM
IE-Take2.cap WHERE DstPort=80 ORDER BY DateTime ASC" -headers:ON
```

Command to query Netmon and find frame numbers based on specific text in payload

```
LogParser -fMode:TCPIP -rtp:-1 "SELECT Frame, Payload INTO 3dvia.txt FROM 3dvia.cap
WHERE DstPort=80 AND Payload LIKE '%ppContent%' " -headers:ON
```

Using System.NET Tracing to debug Network issues

http://blogs.msdn.com/dgorti/archive/2005/09/18/471003.aspx

List of figures and diagrams

Index