|  |
| --- |
| Microsoft Corporation |
| TSL Power Tools |
| Routines for web and load testing with Visual Studio 2010 |

|  |
| --- |
| Microsoft Testing Services  12/27/2011 |

# Code Snippets

Code snippets are pieces of code that can be automatically inserted when writing applications. They add the defined code and then expose the variables in the code that should be renamed. *The code snippets included in this tool are all specific to c#.*

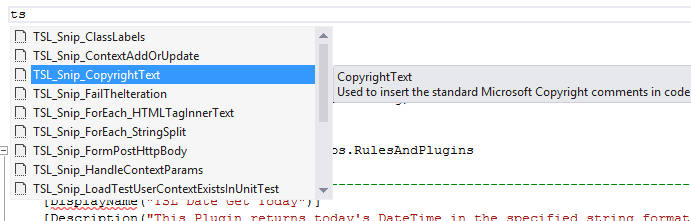
## How to Use Snippets

To Use, copy the .SNIPPET files into the following directory:

**Documents\Visual Studio 2010\Code Snippets\Visual C#\My Code Snippets**

When you are writing code, these will now be available as auto-complete items.

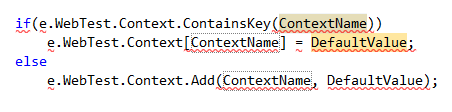
* Start your typing (anywhere in the code editor) with “MTS” (upper or lower case). A list box appears showing the snippets available that start with the typed text. Each snippet has a description.
* Highlight the snippet you want to use (you can click it with the mouse or use the arrow keys to scroll through. NOTE: All of these snippets start with “TSL\_SNIP\_”. This makes it easy to find all of the snippets included in this toolkit:



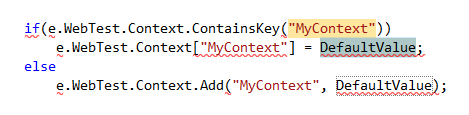
* Press “tab”. The list box will be replaced by full snippet name. It will also be underlined, letting you know that the code is not complete.



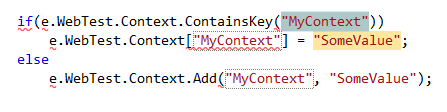
* Press “tab” a second time. The actual code will appear, with certain items highlighted. In the below example, the word “ContextName” is in tan and the word “DefaultValue” is in yellow:



* Type in a value you wish to use and then press tab again (in this example, I typed in “MyContext”). The value that was highlighted gets replaced, AS WELL AS any other values that are the same (in this case, two other items were changed). Also, the gray highlight will move to the next item you can replace.



* Continue this until you have filled in all of the items that are highlighted.



The highlighter will continue to loop through all of the marked items until you stop pressing “tab”. This gives you achance to go back and correct any typos.

NOTE: Once you start typing elsewhere, you no longer have the option of tabbing through the items. They become just plain code at that point.

The following snippets are included in this version:

**TSL\_CodePieces.snippet**

TSL\_Snip\_ContextAddOrUpdate

TSL\_Snip\_HandleContextParams

TSL\_Snip\_FailTheIteration

TSL\_Snip\_ForEach\_HTMLTagInnerText

TSL\_Snip\_ForEach\_StringSplit

TSL\_Snip\_FormPostHttpBody

TSL\_Snip\_LoadTestUserContextExistsInUnitTest

TSL\_Snip\_LoadTestUserContextExistsInWebtest

SQLConnectionWithDataReader

**TSL\_Declarations.snippet**

TSL\_Snip\_CopyrightText

TSL\_Snip\_ClassLabels

TSL\_Snip\_MethodLabels

TSL\_Snip\_ProjectComments

TSL\_Snip\_PropertyLabels\_bool

TSL\_Snip\_PropertyLabels\_int

TSL\_Snip\_PropertyLabels\_string

## ****Snippet Descriptions****

**TSL\_Snip\_ContextAddOrUpdate**

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Copyright (c) Microsoft. All rights reserved.

// THIS CODE IS PROVIDED \*AS IS\* WITHOUT WARRANTY OF

// ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING ANY

// IMPLIED WARRANTIES OF FITNESS FOR A PARTICULAR

// PURPOSE, MERCHANTABILITY, OR NON-INFRINGEMENT.

//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Use this on all code that we give to customers. This is a standard Microsoft boilerplate disclaimer

**TSL\_Snip\_ContextAddOrUpdate**

if(e.WebTest.Context.ContainsKey(ContextName))

e.WebTest.Context[ContextName] = DefaultValue;

else

e.WebTest.Context.Add(ContextName, DefaultValue);

Use this when you are adding or updating any context parameter. It performs a check for the existence of the context to ensure you are not trying to either set a non-existent param or not trying to create a param that does already exist. You should use this validation in your code ANY TIME you are setting values or creating new context params.

**TSL\_Snip\_FailTheIteration**

{

e.WebTest.AddCommentToResult("ReasonForFailure");

e.WebTest.Outcome = Outcome.Fail;

e.WebTest.Stop();

}

Use this when you want to add an iteration failure to a plugin. Web and load test plugins do not automatically stop or fail tests when they fail, so this extra code is needed. One variation to consider allows you to specify whether a failure should stop the iteration and/or fail the iteration. You must define the two Booleans (bFail and bStop) prior to this piece of code. This variation is not included as a snippet.

**TSL\_Snip\_** **ForEach\_HTMLTagInnerText**

HtmlTagInnerTextParser tagParser = new HtmlTagInnerTextParser(e.Response.BodyString);

foreach (HtmlTagInnerText tInner in tagParser.GetInnerTextForHtmlTags("select", "Name", "Value", true, false, true))

{

}

This sets up the code needed to walk through the inner text of the defined HTML tags in the response from a web request. The example below shows how you can use the **HtmlTagInnerTextParser** built into the web testing to search inside a **select** tag and randomly pull an option out of that text:

Sample HTML Code to parse

//<select name="ctl00$ContentPlaceHolder1$statementDateDropDown"

id="ctl00\_ContentPlaceHolder1\_statementDateDropDown"

//class="inputfieldsDropdown" minPayment="10000" name address1 address2 city

style="width:175px;">

// <option selected="selected" value="15/06/2010"></option>

// <option value="16/05/2010"></option>

// <option value="15/04/2010"></option>

// <option value="16/03/2010"></option>

// </select>

HtmlTagInnerTextParser tagParser = new HtmlTagInnerTextParser(sStr);

foreach (HtmlTagInnerText tInner in tagParser.GetInnerTextForHtmlTags("select", sAttributeName, sAttributeValue, true, false, true))

{

The code needed to walk the HTML Body from the response and get the random value

// We should only get one tInner that matches criteria

sInner = tInner.Value.Split(new string[] { "value=\"" },

StringSplitOptions.RemoveEmptyEntries);

Random rand = new Random();

int x = rand.Next(sInner.GetUpperBound(0));

string sValue = sInner[x];

e.WebTest.Context[this.ContextParameterName] = sValue.Substring(0,

sValue.IndexOf("\"></option"));

}

**TSL\_Snip\_ForEach\_StringSplit**

foreach (string str in mainString.Split(new string[] { ";" }, StringSplitOptions.RemoveEmptyEntries))

{

}

This code is used to parse through a string containing substrings separated by semicolons. A good example of using this is to parse a plugin property where you wish to allow multiple entries. The following example demonstrates this technique:

Public property where you can enter more than one value for a single property

[DisplayName("Context Names For GUIDS")]

[Description("The name(s) of the context parameter(s) used to hold the

values. Separate each name with a ;")]

[DefaultValue("prmGuid1;prmGuid2")]

public string ContextNames { get; set; }

Using the loop to set or add a context for each substring in the collection

foreach (string str in ContextNames.Split(new string[] { ";" },

StringSplitOptions.RemoveEmptyEntries))

{

if (e.WebTest.Context.ContainsKey(str))

e.WebTest.Context[str] = Guid.NewGuid().ToString();

else

e.WebTest.Context.Add(str, Guid.NewGuid().ToString());

}

**TSL\_Snip\_** **FormPostHttpBody**

FormPostHttpBody fphb = new FormPostHttpBody();

fphb = (FormPostHttpBody)e.Request.Body;

{

// --- Add A Value

//fphb.FormPostParameters.Add(sTemp, "on");

// --- Walk all values

//foreach (FormPostParameter fpp in fphb.FormPostParameters)

//{

// // Do some action

//}

// --- Remove Specific Params

// We need to clear all params which have our SubstringToSearchFor

//for (int x = fphb.FormPostParameters.Count - 1; x < 0; x--)

//{

// if (fphb.FormPostParameters[x].Name.Contains(str))

// fphb.FormPostParameters.RemoveAt(x);

//}

}

e.Request.Body = fphb;

The above snippet sets up the necessary code to manipulate a form post body. The webtesting namespace does not allow you to directly manipulate the body, so you need to create a temporary body, set it to the request’s body, then change the temporary one. When you are done, you assign the temporary one back to the request’s body.

Included in this snippet are examples of a couple of things you might want to do with the body. Uncomment whatever one(s) you need.

**TSL\_Snip\_** **HandleContextParams**

// NOTE: This routine is what handles context params passed in as values in the plugin

string clean"MyString" = "";

if ("MyString".Contains("{{"))

{

"MyString" = "MyString".Replace("{{", "");

"MyString" = "MyString".Replace("}}", "");

clean"MyString" = e.WebTest.Context["MyString"].ToString();

}

else

clean"MyString" = "MyString";

This code allows your plugins to accept context parameters as values passed into the public properties of the plugin. In the snippet code, the value “**MyString**” is the name of the property. The snippet creates a new string called **clean”MyString**” which will hold the actual value. When adding the rest of the code for the plugin, you will need to reference the **clean”MyString”** for the value. The snippet is written this way so you can add a single name to the snippet and it will change every item in the snippet.

**TSL\_Snip\_** **LoadTestUserContextExistsInUnitTest**

if (TestContext.Properties.Contains("$LoadTestUserContext")) //running as load test

{

// add your code here, or use one of the below lines

//int iAgentId = Convert.ToInt32(TestContext.Properties["$AgentId"]);

//int iAgentCount = Convert.ToInt32(TestContext.Properties["$AgentCount"]);

}

**TSL\_Snip\_** **LoadTestUserContextExistsInWebtest**

if (e.WebTest.Context.ContainsKey("$LoadTestUserContext")) //running as load test

{

// add your code here, or use one of the below lines

//int iAgentId = Convert.ToInt32(e.WebTest.Context["$AgentId"]);

}

Both of these routines will look for a load test context and, if present, will retrieve the value. The only difference in the routines is the syntax of the context itself.

**TSL\_Snip\_** **ProjectComments**

//===========================================================================

// "somefile.cs"

// Written by "Your Name Here"

// created on "1/1/01"

//

// add comments here

//===========================================================================

This snippet adds the proper code for creating header comments for your projects

**TSL\_Snip\_** **MethodLabels**

[System.ComponentModel.DisplayName("ContextName")]

[System.ComponentModel.Description("ContextDescription")]

**TSL\_Snip\_** **PropertyLabels\_bool**

[System.ComponentModel.DisplayName("ContextName")]

[System.ComponentModel.Description("ContextDescription")]

[System.ComponentModel.DefaultValue(true)]

public bool bBoolName { get; set; }

**TSL\_Snip\_** **PropertyLabels\_int**

[System.ComponentModel.DisplayName("ContextName")]

[System.ComponentModel.Description("ContextDescription")]

[System.ComponentModel.DefaultValue(0)]

public int iIntName { get; set; }

**TSL\_Snip\_** **PropertyLabels\_string**

[System.ComponentModel.DisplayName("ContextName")]

[System.ComponentModel.Description("ContextDescription")]

[System.ComponentModel.DefaultValue("DefaultValue")]

public string sStringName { get; set; }

These snippets allow you to easily add public properties to plugins and to add the proper labels to a method. Refer to the section on plugin templates for an explanation of the fields and how to populate them.

**TSL\_Snip\_** **SQLConnectionWithDataReader**

string sqlCmd = "SELECT \* FROM dbo.TableName";

string sqlConn = "Data Source=Server;Integrated Security=SSPI;Initial Catalog=DataBaseName";

SqlConnection \_conn = new SqlConnection(sqlConn);

\_conn.Open();

SqlCommand \_cmd = new SqlCommand(sqlCmd, \_conn);

SqlDataReader \_rdr = \_cmd.ExecuteReader();

string ColumnName1 = "";

string ColumnName2 = "";

while (\_rdr.Read())

{

ColumnName1 = \_rdr.GetString(0);

ColumnName2 = \_rdr.GetString(1);

}

\_rdr.Close();

This snippet adds the code necessary to make a SQL Connection and read data from the connection. It is generic so it can be used in any class.