

TelQA: How to Create CodeScripts



Table of Contents

1.0 Introduction.....	1
2.0 CodeScript called from SmartScript.....	2
3.0 CodeScript generated from SmartScript.....	7

1.0 Introduction

SmartScripts are generally the preferred scripting method when using TelQA Test, as these are quick to generate and edit, do not require C# or VB.NET programming expertise, and provide a powerful graphical user interface geared specifically toward the development of web and terminal based scripts. However, there are situations where SmartScript capabilities are insufficient for particular scripting needs, and in these cases a C# or VB.NET CodeScript may be used. These allow access to the full power of .NET programming, including the ability to use functionality provided by both Microsoft and third-party libraries. Note that CodeScripts are not supported in TelQA Test Lite.

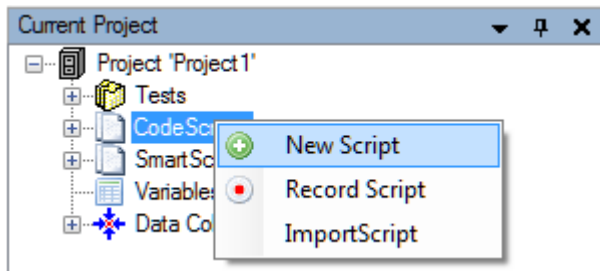
SmartScripts and CodeScripts may be used in combination within a test. It may be that a CodeScript is only necessary for implementing one portion of a test and, in this case, the bulk of the test may be implemented using SmartScripts, with a CodeScript executed for one part of the test sequence, or called as a sub-script from one or more SmartScripts.

The following sections describe how to create and use CodeScripts within a test.

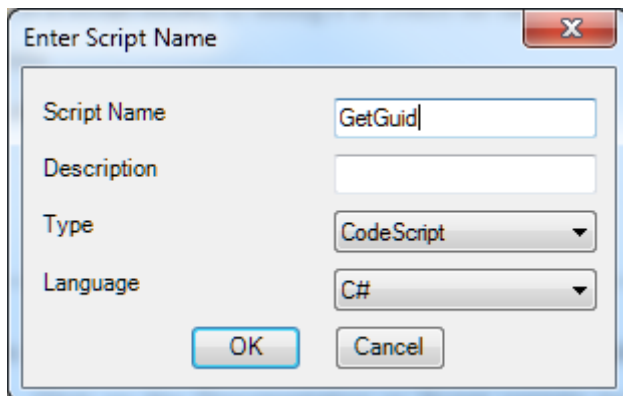
2.0 CodeScript called from SmartScript

This example will show a very simple CodeScript which is used to obtain some data which cannot be obtained directly using a SmartScript. We will assume that the CodeScript needs to obtain a unique string for each VU instance, and will generate a GUID for this purpose using System.Guid.NewGuid(). This value will be returned to the SmartScript using a TelQA Test variable called "guid".

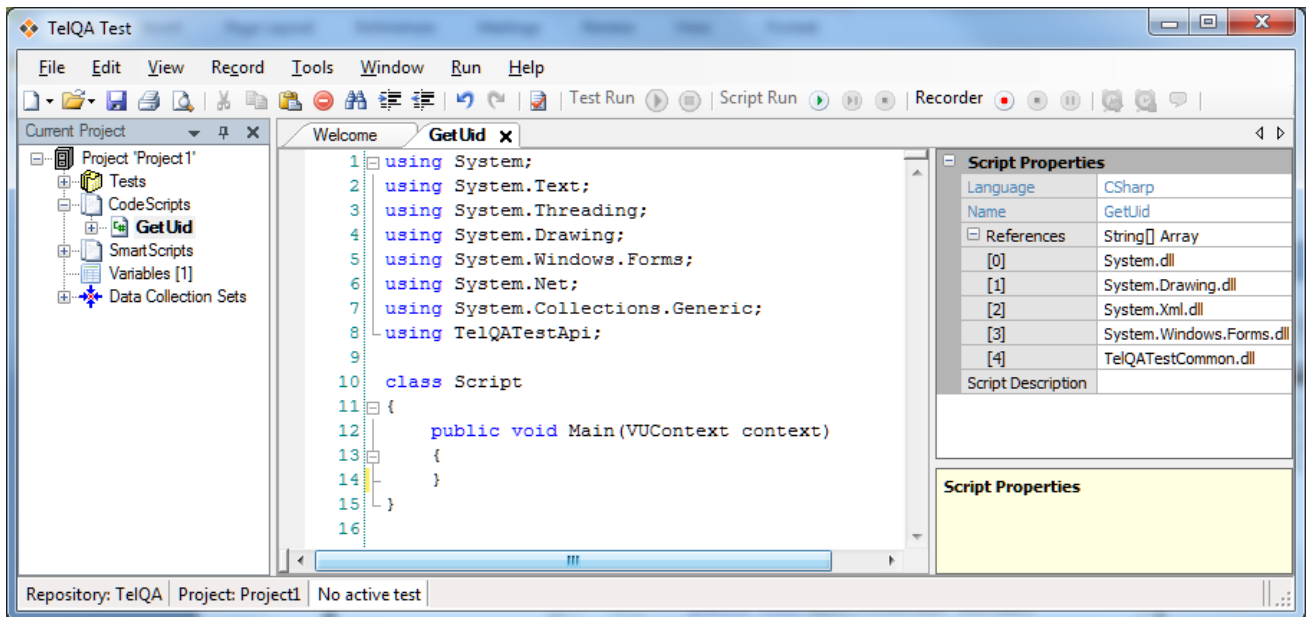
If not already open, open the Project window using View>Project. Right click the CodeScripts node of the project tree and select the New Script option:



Enter "GetGuid" as the script name, the language as C# and click OK:



A new script node will be added under the CodeScripts tree node, and a skeleton script will be displayed in the document window:



Note the following points about the newly created script:

- The script consists of a class **Script** containing a method called **Main** with a single argument of type **VUContext**. The **Main** function will be called by the TelQA Test runtime test executer within a separate thread for each VU. **VUContext** may be used by the script to reference context-specific data and methods using the TelQA Test API. The **Main** method returns no data, but may set one or more variables in order to pass information to other scripts.
- The References property is a list of DLLs which the script references (equivalent to the project references in a Visual Studio project). Scripts created by TelQA Test will include a default set of references, but you may need to add extra ones when calling additional .NET or third-party classes. Note that any such referenced DLLs should be located either in the GAC (this applies to all standard .NET DLLs) or in the TelQA Test installation folder.
- A CodeScript may call any .NET method within reason, although it should generally avoid methods which involve user I/O. Note that runtime values may be passed to a test using Test Parameters, and user feedback may be obtained using the **VUContext.MessageBox** method.
- If an error is encountered in a CodeScript, it may be handled in one of two ways. If the error is non-fatal and it makes sense for the script to continue, the error may be reported in the Event Log using **VUContext.WriteError** or **VUContext.WriteException** (with the abort flag set to false). If the error is such that the script cannot continue, **VUContext.WriteException** should be called with the abort flag set to true. Note that if an unhandled exception occurs within a script, this will be recorded in the Event Log and the VU will be aborted.

For our example, insert the following line within the Main method:

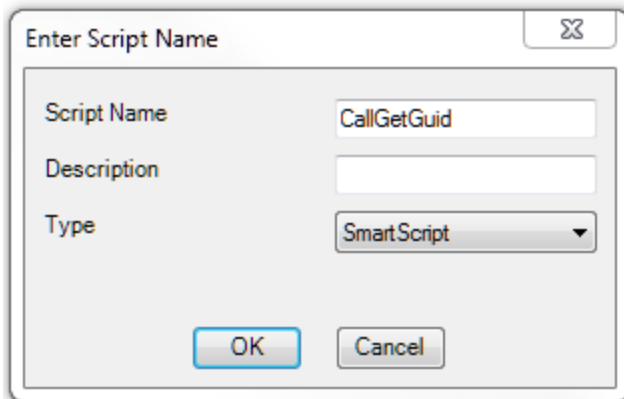
```
context.SetVariable("guid", Guid.NewGuid().ToString());
```

Note that, as you type, the IntelliPrompt system displays popups to assist in the editing process. Note also that the Errors window is updated dynamically to show any syntax errors within the script.

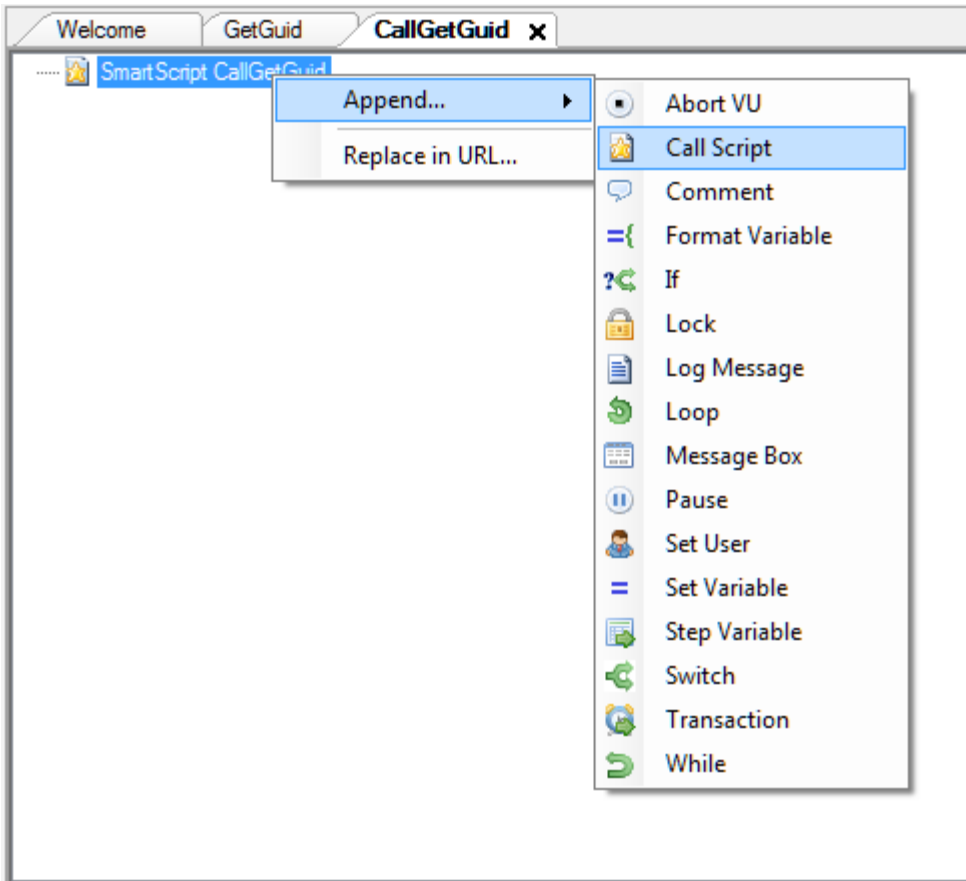
The above statement will create a new GUID, convert it to a string, and store its value in a new variable called "guid". This variable will be available to any other script run subsequently within the current test for the same VU. If the test runs several VUs, then each VU will get a unique value for "guid".

To check that the script runs successfully, right-click the GetGuid script node in the project tree and select "Run Script". This causes a temporary test to be generated, consisting of a single VU running the script for one iteration with no startup delay. The script should run very quickly and produce an entry under the Script Runs node in the project tree. Double-click the "Result View" node; this will display the Event Log for the script run, which should contain no errors. Note also that after the script has been run the Variables window is displayed, showing the value of all variables at the end of the run, including that of the "guid" variable.

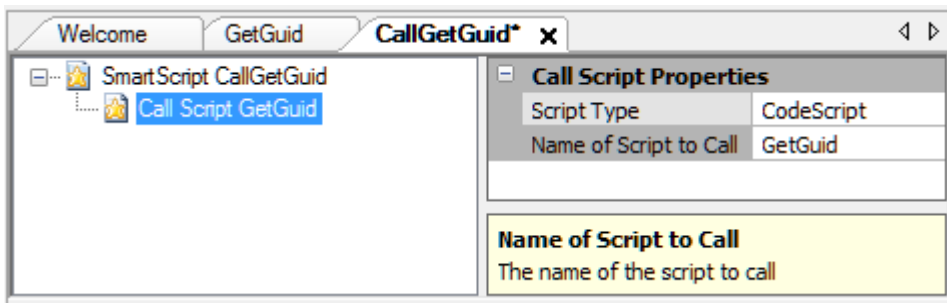
We will now create a SmartScript which will call the CodeScript to obtain the GUID and log its value in the event log. Right click the SmartScripts node of the project tree and select the New Script option. Enter "CallGetGuid" as the script name and click OK:



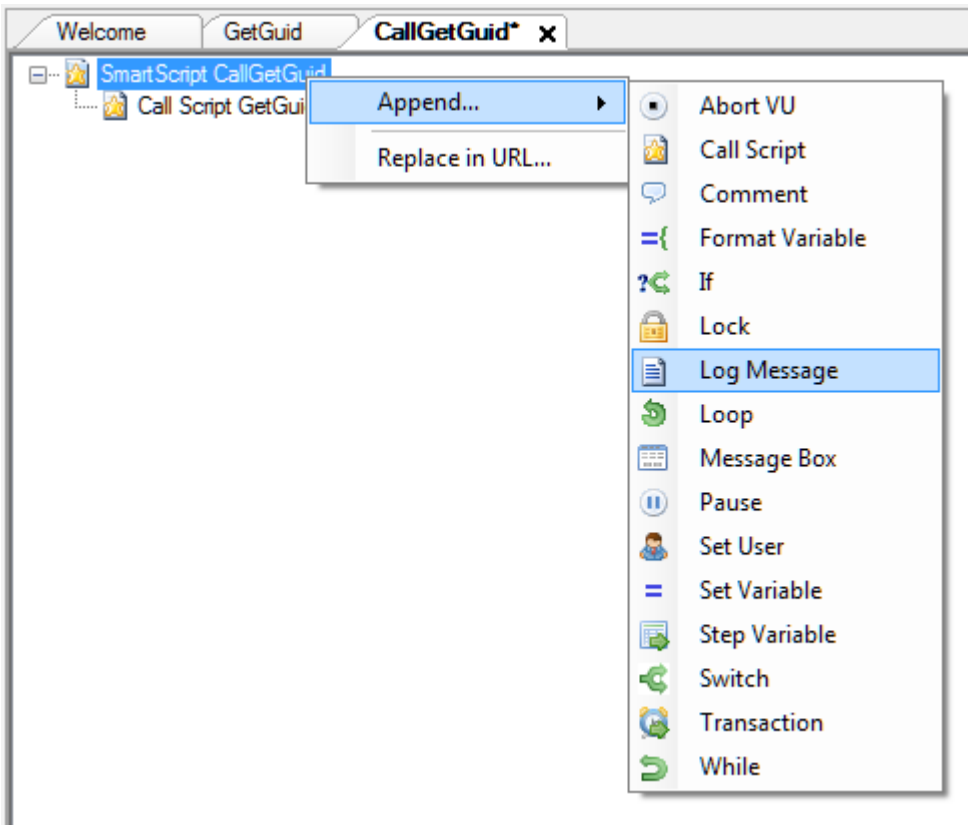
A new script node will be added under the SmartScripts tree node, and a skeleton script will be displayed in the document window. Right-click the script node and select "Append..." then "Call Script":



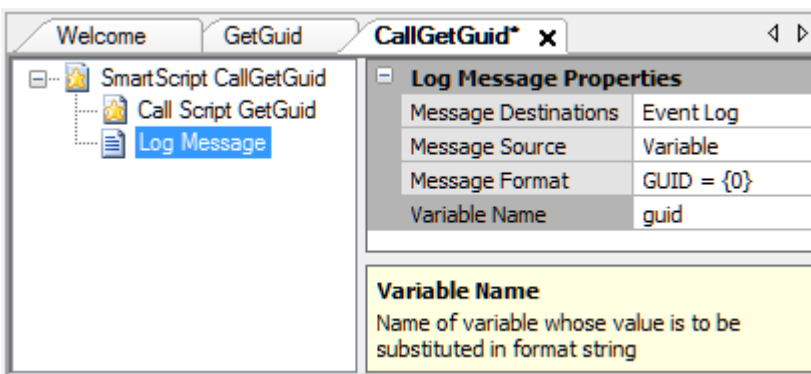
Select CodeScript as the Script Type, and GetGuid as the name of the script to call:



Now right-click the script node again and select "Append..." then "Log Message":



Leave the Message Destination as Event Log and select Variable as the Message Source. Edit the Message Format field to set it as "GUID = {0}" and enter "guid" as the Variable Name. Note that in this case the variable name "guid" will not be included in the drop-down list as it is set solely within the CodeScript and the list only contains variables referenced previously within the SmartScript or defined as project variables:

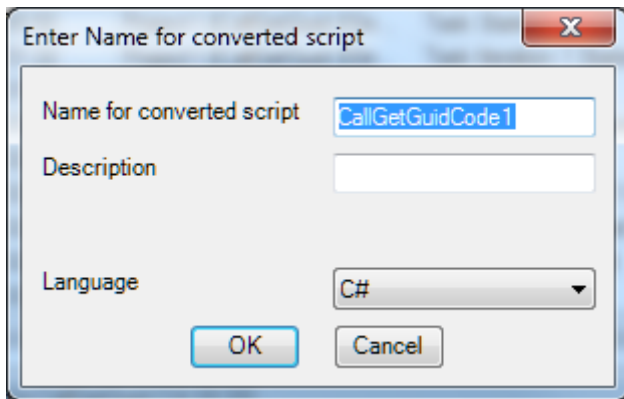


To check that the script runs successfully, right-click the CallGetGuid script node in the project tree and select "Run Script". The script should run very quickly and produce an entry under the Script Runs node in the project tree. Double-click the "Result View" node; this will display the Event Log for the script run, which should contain a User Message event including the GUID value.

3.0 CodeScript generated from SmartScript

As well as creating a CodeScript from scratch, you can convert an existing SmartScript into a CodeScript. This may be useful if you need to perform some special processing which cannot easily be achieved in a SmartScript. To illustrate this we will convert the SmartScript created above into a CodeScript.

Right-click the CallGetGuid script node in the project tree and select "Convert to CodeScript". Leave the defaulted script name and the language as C# and click OK:



This will convert the script and display it in the document pane:

```
1 // Script converted from SmartScript CallGetGuid on 23-Jul-2010 14:53:41
2
3 using System;
4 using System.Text;
5 using System.Threading;
6 using System.Drawing;
7 using System.Windows.Forms;
8 using System.Net;
9 using System.Collections.Generic;
10 using TelQATestApi;
11
12 class Script
13 {
14     public void Main(VUContext context)
15     {
16         System.String text = null;
17
18         context.TraceScript("Start SmartScript CallGetGuidCode1", 0, 1);
19
20         // Call Script GetGuid
21         context.TraceScript("Call Script GetGuid");
22         context.CallScript("GetGuid", false);
23
24         // Log guid to Event Log
25         text = String.Format("GUID = {0}", context.GetVariableValue("guid"));
26         context.TraceScript(String.Format("Log to EventLog: {0}", text));
27         context.WriteUserMessage(text);
28
29         context.TraceScript("End SmartScript CallGetGuidCode1", -1, 0);
30     }
31 }
32
```

Note that many of the code steps involve calling member functions of the VUContext class. This is the primary interface between CodeScripts and the TelQA Test runtime system and is documented in Help>TelQA Test API.

To check that the script runs successfully, right-click the CallGetGuidCode1 script node in the project tree and select "Run Script". The script should run and produce the same results as the SmartScript version.