Using Macro Scheduler For Load Testing

Using Macro Scheduler for Automated Load Testing

This document describes how Macro Scheduler can be used for load testing other software applications.

1. Simulating a Multi-User Environment
2. Synchronising Scripts
3. Measuring Elapsed Time
4. Responding to Events
5. Recording and Analysing the Results
6. Logging to a Database

Simulating a Multi-User Environment

This has been done a number of ways. The most common approaches are to use Windows Terminal Server or Microsoft Virtual PC. Using these solutions one physical machine can be divided into several virtual PCs. Using a number of physical machines multiplexed into several virtual PCs it is easy to build an environment of multiple virtual workstations.

Each virtual workstation must have access to the software that is being tested and a copy of Macro Scheduler running on it.

Synchronising Scripts

It is often necessary to ensure that all workstations begin testing at the same time. This can be done a number of ways and a couple of methods will be described here.

Method 1: The Controlling Script Semaphore Method

The simplest approach is simply to have a controlling script, running on a master workstation, which sets a flag to indicate that the testing scripts should start. This flag is usually just a value in an INI file, which is stored on a network share available to all testing scripts.

A very simple start script might simply look like this:

```
Let>inifile=\\testserver\mainshare\tests.ini
EditIniFile>inifile,main,run,1
```

The INI file would look like this:

```
[main]
run=0
```

When the start script is run it sets the run flag in the INI file to 1.

The test scripts will be set to wait for this flag to be 1 before performing the test:

```
// Wait for run flag
Let>inifile=\\testserver\mainshare\tests.ini
ReadIniFile>inifile,main,run,runval
Repeat>runval
  ReadIniFile>inifile,main,run,runval
Until>runval=1

// run the test …
```

Each test script would be set to run on start up so that when the test stations are booted (or Macro Scheduler is started on those machines) they immediately start and wait to be told to perform the test. This avoids any manual process on the test machines.

If we add different tests we can add appropriate flags to the INI flag and add a corresponding control script:

```
[main]
run=0

[invoiceentry]
run=0

[reporting]
run=0
```

Now we can start different types of test by running the appropriate control script.

We will also want to add a control script to reset the run flag for later tests.

Method 2: msNet Server Add-in

Method 2 utilises the msNet server add in which allows macros to be started across the network. msNet adds a small server component to each Macro Scheduler workstation so that a request can be made remotely to run a script. Scripts can be started from a web browser, command line tool, or GUI.

A broadcast request can be made so that the same script is run on all msNet enabled machines at once.

For more information on msNet see:
http://www.mjtnet.com/msnet.htm

Measuring Elapsed Time

Of most importance during load testing is the ability to measure elapsed time to determine how long certain processes take. There are a number of ways to do this with Macro Scheduler.

TimeStamp

The TimeStamp command outputs an accurate time, with milliseconds, and a message, to a log file. This can be used to log events during the test script. When the test is finished the log file can be imported into Excel and analysed.

Timer

Sometimes it may be preferable to calculate elapsed time in the script directly and output the actual elapsed time. This cuts down on the analysis needed later.

Often we'll use a combination of these two techniques. The following script code determines how long a logon took to complete and outputs the results to the log file:

```
// Need these lines at start of script
VBSTART
VBEND

// Wait's for logon window
WaitWindowOpen>OneWorld Sign On*

// Perform logon
SetFocus>OneWorld Sign On*
Press Tab * 5
Send>%UserID%
Press Tab
Send>%UserPWD%
Press Tab
Send>%Environment%

// Output timestamp and message
TimeStamp>%logfile%,About to Log on

// Get Start Time
VBEval>Timer,StartTime

// Hit log on button
Press ALT
Send>o
Release ALT

// Wait for login to complete
WaitWindowOpen>J.D. Edwards OneWorld Explorer*
// Calculate elapsed time
VBEval>Timer-%StartTime%,ElapsedTime

// Output elapsed time
TimeStamp>%logfile%,Elapsed: %ElapsedTime%
```

So here we have used TimeStamp and Timer to output critical time information about how long the log on process took. The log file can be imported into Excel and analysed.

For more control over the format of the output use the WriteLn command. We can create a CSV file by separating key data with semi-colons:

```
WriteLn>logfile,r,Start Log on;%ElapsedTime%
…
WriteLn>logfile,r,Done Log on;%ElapsedTime%
```

Responding to Events

Most of the time we want to determine how long a process took. The majority of the time this is as simple as waiting for a window to appear. E.g. in the log in example above we enter the log in information, send alt-o to ok the log in and then wait for the main application window to appear. We then know the log in has been successful and we can calculate the elapsed time and log the results.

Sometimes it is not possible to determine progress by looking at windows and we need to use other approaches. Amongst other things Macro Scheduler has functions to wait for the cursor to change; get object text; wait for pixels to change colour; get a checksum for a rectangular portion of the screen and wait for that checksum to appear; wait for windows to change; check for the existence of files. Check out the command reference that comes with Macro Scheduler for a full list of commands. There is always a way that one or other of these commands can be used to determine a required outcome.

For example, the following script snippet waits for the status bar to say ‘Done’:

```
SetFocus>Internet Explorer*
Press F5
GetActiveWindow>title,X,Y
Let>result=
Label>waitDone
  GetControlText>title,msctls_statusbar32,1,result
  If>result<>Done,waitDone
//complete - output elapsed time …
```

This example waits for the first status bar in Internet Explorer to say Done. In fact if we were really automating Internet Explorer we’d probably use it’s Busy property directly rather than look for the status bar text. But the example demonstrates a method which can be used in any application and any control. It doesn’t have to be a status bar object – we could be waiting for a button to change text for example.

Version 8.0 of Macro Scheduler also includes OnEvent event handlers. These allow us to respond to events that may happen at indeterminable times. These could be error messages for example. We can define an OnEvent handler for an error message and log when that appears.

Another approach is to use OnEvent to log performance data and therefore not have to add logging and time calculations to the body of the actual script. This makes it easy to add performance logging to existing automation scripts easily.

Recording and Analysing the Results

We've already discussed how we can record the performance data in the previous section. TimeStamp is designed specifically for outputting script performance data. Alternatively we can calculate elapsed time and output it using WriteLn. Or we can use a combination of TimeStamp and elapsed time to provide an output file with millisecond accurate time and elapsed time. The output file(s) can then be imported into Excel and analysed.

It is often preferable to output performance data directly to a database. This is faster and ensures all performance data is in one place and allows more advanced analysis using SQL and graphical query and reporting tools.

Logging to a Database

A better alternative to outputting to a CSV or text file is to output performance data directly to a database. This can speed up the process and make analysis easier. We can easily write to a database using VBScript. E.g.:

```
//Add to start of script – this opens a DB connection
//and contains the function to output performance data
VBSTART
   'open the database connection
   set DB = CreateObject("ADODB.Connection")
   'next line can be a data source name or connection string
   DB.Open "DataSourceName"
Sub LogOutput(scriptId,elapsed, message)
   SQLString = "INSERT INTO tblResults (scriptId, elapsed, msg)
VALUES ('" & scriptId & "', '" & elapsed & "','" & message & "')"
   DB.Execute(SQLString)
End Sub
VBEND

//we would log data like so:
VBRun>LogOutput,1,%elapsed%,Done Logon Screen 1
```

This example simply outputs three fields – a script ID, elapsed time and a message. This can easily be modified to output any amount of information to suit the needs in hand and the structure of the database.