Managing QA Partner Test Results in Lucent's BUSTER™ Test Management System

Segue's Customer Newsletter, segue@work, Spring 1998

Testing in a heterogeneous environment can present the test engineer with a unique set of challenges. These challenges can be further complicated if the test organization uses tools from different vendors that run on multiple hardware platforms and operating systems. Normally, tools from different and competing vendors are very loosely integrated or not integrated at all. This article presents a solution to the problem of dealing with a test management system, called BUSTER, hosted on a UNIX platform and with a Windows 95 version of the QA Partner automated test tool.

BUSTER is a test management system originally developed by AT&T which is now maintained and supported by Lucent Technologies. It was developed specifically for test scripts written in some flavor of UNIX shell for the purpose of testing server application software. It is not uncommon to encounter organizations that have standardized on the BUSTER test management system that are UNIX oriented. However, many of these organizations are beginning to develop desktop applications on PCs instead of high-cost UNIX workstations; while continuing to develop server software on UNIX. This is just such an environment where it was necessary to enable QA Partner to generate BUSTER formatted results.

Before attempting a solution, a self-imposed series of informal constraints or requirements were established. The following is a list of these constraints:

- Any code needed to support the capture of these results must be non-intrusive. That is, the test engineer should not have to develop a lot of extra test script code to capture the results.
- The solution must not require the test engineer to develop or execute elaborate conversion scripts on the UNIX platform to format the results. The solution should be PC-based.
- Existing Windows 95/Windows NT networking features must be used to transport the results from the PC to the UNIX machine running the BUSTER system.

Since one of the constraints is for the solution to be non-intrusive, this immediately eliminated the obvious solution which is to code print statements in selected areas of each test case where failures were detected. This solution would also require a lot of additional test script maintenance if the BUSTER results database format would change in future product releases.

QA Partner was able to meet these requirements through its powerful and feature-rich object-oriented 4Test script language. This opened the way to a simple and elegant solution by exploiting the ability to override QA Partner's default recovery system. For more details on the default recovery system refer to the QA Partner User's Guide Chapter 14, "Customizing QA Partner." The default recovery system consists of the following built-in functions: DefaultBaseState, DefaultScriptEnter, DefaultTestCaseEnter, DefaultTestCaseExit, and DefaultScriptExit. These functions are called automatically during execution of your test script. These functions can be overridden by re-naming and coding them in an include file. By renaming these functions as ScriptEnter, TestCaseEnter, TestCaseExit, and ScriptExit, they will be executed instead of the default versions.

Since the solution described in this article is to create BUSTER test results from a QA Partner test script, the format of a BUSTER runinfo record is listed here. BUSTER runinfo records contain 15 fields separated by a caret (^) as follows:

Id	test identification from the ID section of a test script.
Name	test name.
Type	type of test session: official (O), modified test (M), unofficial – no checksum (U), and debug (D).
Session	unique test session id obtained from environment variables LABNAME and TMSTMP.
Sdate	start date of the test.
Stime	start time of the test.
Brtime	elapsed real time of the entire test execution in decimal minutes.

Prtime	elapsed real time of the PROCEDURE section of a test execution.
Pass	total passing testcases.
Fail	total failing testcases.
Inc	total inconclusive testcases.
Nr	number of testcases not run.
Config	value of environment variable CONFIG.
Usercom	comments from brun(1B) or the brun functions.
Runuser	reserved for individual project use.

Along with the default recovery system, this solution also makes use of the Compiler Constants feature of the Runtime Option menu. Compiler Constants provide the information to simulate the UNIX environment variable CONFIG and provides the path name for the creation of the result file. The entries needed in the Compiler Constants for this application are:

Constant Name	**Value**
CONFIG	"Version.inc"

Version.inc is a 4Test include file containing the following entries:

```
STRING sConfig  = "TRM,Release1.0"
STRING sDir     = "C:\QAPResults"
```

The philosophy used to design these functions was simply to build a runinfo string containing all the caret delimited fields and then write this string to a uniquely named file. The implementation for each function will now be described.

The ScriptEnter function contains code to perform the following:

- Get the current date and time and format a timestamp.
- Create the result folder as specified in the supplied Compiler Option.
- Verify that the test script name is 12 character or less and force all uppercase characters.
- Stuff the runinfo string with the id and name fields derived from the 4Test Script name.
- Stuff the runinfo string with the type field assigned the value "O"
- Stuff the runinfo string with the session field assigned the result of the GetMachineName () function concatenated with a formatted date/time.
- Stuff the runinfo string with the sdate and stime fields derived from the formatted timestamp.

The TestCaseEnter function is not used for this solution and is commented out to indicate that the DefaultTestCaseEnter function is used.

The TestCaseExit function is not as straightforward. There needed to be a way to account for data-driven test cases. That is, the function needed to be able to distinguish between test cases executed multiple times with different data from a single invocation of a test case. The TestCaseExit function maintains a count of each execution of a data driven test case and displays failed test cases in the usercom field in the following form: TestCaseName[i]:n error(s)! Where i is this test case iteration and n is the count of errors. This function builds a string with the above information which is used later on by the ScriptExit function.

The ScriptExit function contains code to perform the following:

- Get the script elapsed time and format the result in decimal minutes.
- Stuff the runinfo string with brtime and prtime fields derived from the elapsed time.

- Stuff the runinfo string with pass, fail, inc and nr fields computed in the TestCaseExit function.
- Stuff the runinfo string with config field whose value is taken from the sConfig variable declared in the Version.inc file.
- Stuff the runinfo string with usercom field whose value is taken from a string created by the TestCaseExit function.
- Stuff the runinfo string with runuser field whose value is "QAP" for this example.
- Write the runinfo string to the output file.

However, in order for BUSTER to be able to accept any results from QA Partner, a BUSTER test script must be created. This test script is not executed but is merely a "place holder" so that the QA Partner results can be associated with a test script in BUSTER. Note that the BUSTER test script maintains the number of test cases using the line: `COUNT: 4`, this count must also be included in the QA Partner test script. Below is a sample BUSTER test script for a QA Partner script called TRMGUI_TRM.t:

```
ID: TRMGUI_TRM.t
ORIGIN: TRM
TYPE: m
OBJECT: TRM Main Window
CONTACT: sczepura
DOC: mfrtrmui.fm
KEYWORDS: HIGH TRM GUI regression
PACKAGE: TRM Release0.5 Release1.0

PURPOSE:
TC. Window Description - Admin
TC. Window Description - Operator
TC. Window Description - Regular
TC. Window Controls

REQT: TRMGUI.TRM
      TRMGUI.MSG

METHOD:
LIBRARY: lib/platform lib/common/manualchk
SHELL:
SCONFIG:
HCONFIG:
COMMENT: Automated using QA Partner or tested manually on GUI.
COUNT: 4
STIME:
PTIME:
CTIME:
SETUP:
PROCEDURE:
$TESTROOT/lib/platform/GUI_Manualchk 4
# QA Partner script: TRMGUI_TRM.t
CLEANUP:
```

The corresponding QA Partner test script would only need to include the following two lines of code to support the generation of BUSTER test results, assuming the include file is named BUSTER.inc.

```
use "BUSTER.inc"

const CNT = 4
```

The following is an example of an actual QA Partner test script results file, in exported format, followed by the QA Partner generated BUSTER formatted runinfo record:

```
basepath  C:\QAPResults
#results   trmgui_trm.res
#fields    TestPlan Script TestCase TestData ErrorCount ErrorText
DateTime Elapsed
#delimiter ,
#quote     " \
#
"","c:\Scripts\trmgui_trm.t","TestWindowDescription","",0,"","1997-08-07
13.07.25","0:00:36"
"","c:\Scripts\trmgui_trm.t","TestWindowControls","\"Admin\"",3,"***
Error: Verify value failed - got \"Trouble Report Management STARS-RE\",
expected \"Work List\"","1997-08-07 13.08.01","0:04:15"
"","c:\Scripts\trmgui_trm.t","TestWindowControls","\"Operator\"",4,"***
Error: Verify value failed - got \"Trouble Report Management STARS-RE\",
expected \"Work List\"","1997-08-07 13.12.16","0:04:08"
"","c:\Scripts\trmgui_trm.t","TestWindowControls","\"Regular\"",3,"***
Error: Verify value failed - got \"Trouble Report Management STARS-RE\",
expected \"Work List\"","1997-08-07 13.16.24","0:06:20"


TRMGUI_TRM.t^TRMGUI_TRM.t^O^local080701307^08/07/97^13:07:25^15.33^15.33^
1^3^0^0^TRM,Release1.0^TestWindowControls[1]: 3 error(s)!
TestWindowControls[2]: 4 error(s)! TestWindowControls[3]: 3 error(s)!^QAP
```

With the BUSTER result file now created on the PC, the next step is to transfer the file to the UNIX machine running the BUSTER test management system. This operation can be performed using FTP, or Windows Explorer if your UNIX file system is remote mounted from your PC. Finally, the results are stored in the BUSTER database using the UNIX command `bstore`.

Biography

Gerard Sczepura is a Consultant currently on assignment with Lucent Technologies. Gerry has almost 20 years experience in the computer field with most of that time spent in software testing and QA. Gerry has been using QA Partner for 2 years on Windows NT and Windows 95 applications. He can be contacted via email at gsczep@epix.net.