Use Your Mainframe
To Test

What's this all about?

I would like to begin the discussion with a quick overview of what the basic idea behind placing tests on your host system is all about. As testers we typically receive software from a development group at the end of the build cycle and then install this software into a given test system. We then run a set of pre-written test cases that exorcise the software in a way that tests the software in a simulated environment. These tests generally take one of 3 forms. 1). We examine manually or programmatically the UI screens that the software produces. 2). We test the objects and methods of those objects in the program by exorcising test code that interacts with the product code. 3.) We do a "System Test" or black box test that places the product in a simulated user environment and then we do the operations that an end user would and verify the results.

Typically we do this in a closed environment. That is one that is completely controlled by us and protected from any outside system or user. This approach provides the benefit of giving us complete control of the tests and ensures the validity of the results. It works and will continue to be the number one way to allow tester access to and control over the test environment. However, this approach does have one drawback to it that I began to appreciate more as my schedule to run tests became less and the complexity of the products grew. The closed system I needed to ensure test integrity became the lock that kept me as the only one that could perform the test plan.

What if we could find a way that would allow us to create tests that worked as well (or better) in an open system environment. This would make the same tests available to anyone and not restrict them to only the tester. No matter haw hard you work in the closed test environment you can never get more work done then you are capable of. But if we could run in an open test environment we would be able to multiplex the testing impact on the product many times over. To use a phrase from Object Oriented Programming we could "write once – use many times".

Such a test system would have to have some common features. 1) It would by definition have to be able to be accessed by anyone who has a desire to use it. 2) It would have to have a common simple language that everyone would be able to use with a minimum of effort. 3) To be useful in any company it would have to be based on equipment that most companies possess. The test area I was concerned about when this idea was born was with IBM systems, so the obvious choice for me was the IBM Mainframe and Midrange AS400 system. However, please note that while this paper will define the specifics for these particular systems the test methodology that it's based on should apply equally well to any other system you may use, such as Unix, HP, IIS, Etc. If it has the ability to edit data, page information, or screen definitions then the potential exists to turn it into a test system. Therefore, most any company is a candidate for using this test methodology.

All of the systems I have used have within them the editors and compilers needed to create user defined Applications. Since the applications can be manipulated by the tester to create any screen or type of data manipulation any customer data transfer can be duplicated. For example, if you have an AS400 that does not have any applications other than the standard ones that came with the OS you can use the editor to create an exact duplicate of any customer application. Even poorly written third-party applications that pop up from time to time can be recreated.

Imagine having a set of libraries that had copies of every customer problem ever reported available to every customer support person. Now imagine a library that sends data in a way that a developer working on a new product you don't even know about yet can test his code without sending it to you. Imagine the team doing the localization of your new product having access to a test library where you wrote tests applications that simulate Arabic or Double Byte languages. These applications can run on English version hosts but send the data as if on a foreign OS. Imagine having test applications that can be written in such a way to be used by Sales or System Engineers to demo your product to customers.

So what do I need to learn to be able to do all of this?

The skills required to create such test libraries may be different than those typically used by testers but anyone with basic programming skills can pick it up quickly. On any given system you will need to learn the editor used, the programming language, and possibly the data stream protocol your system uses. Many systems can use multiple languages such as C or Rexx or SEU. For example, to create these tests on an IBM Mainframe you could use Rexx programming and at least the basics of the 3270 protocol. AS400 users could use CL programming language, the Program Development Manager application, and if desired the 5250 protocol. All of these languages / protocols are not that hard to learn and a fairly large number of test applications can be created by learning only a small subset of the languages.

Ok I think I'm interested but give me some additional benefits to using the host to test.

In addition to the main benefit of testing in an open system by placing a test library on the host, some other significant advantages are:

1) Tests are available at any stage of the product development.
2) The testing is transport independent. If you have a network environment that utilizes several communication protocols you can write a single test that runs on any of them.
3) The tests can be run in localized environments. You can easily write a test that is created to test a foreign language that runs on a US OS, or visa versa.
4) The tests are device independent. Some devices may require a special setup to talk to a test system, but any production device can talk to a host with no configuration.
5) Tests can be made that simulate real customer environments, not simulations of those environments.
6) Customer problems can be retrieved from trace information and turned into test cases quickly.
7) Bug fixes can be verified in house without have to go back to the customer.
8) Performance data can be gathered by using the host resources such as SNMP.
9) Customer Support personnel can use the tests to simulate customer problems.
10) It reduces duplication of effort. In many companies one department may duplicate the same tests as another.
11) Stress testing is almost unlimited. Your host will handle more clients than most any closed test system can dream off.

I can't get anyone to ever try my stuff, how will this help?

One difficulty that always arises when a tester writes tests and then attempts to get others to use them is that they won't do it. No matter how helpful the test may be most people just refuse to use it. I don't have the complete solution for this but I have two ideas that have worked well for me.

My first recommendation is that if the test you wish to get someone to use is harder than getting a pop out of a vending machine it will never be used. The test MUST have almost no setup and run with a simple command line invocation. Anything more is the kiss of death. Test libraries on a Mainframe require nothing more than a standard connection to the host computer and only requires a single command to run, such as test1 (enter). The test itself can be written so that no (or minimal) user involvement is required. Just remember that unless you plan on running the test yourself all the thinking needs to be done while designing the test, not while its being run.

My second recommendation is on how you present your tests to others. For example imagine walking into a developers office and making this statement:

"Hi Bob. I wrote this great new test that I think will help you with the new thing-a-bob".

What are the odds he will use it? Slim probably. What if you change one word to say:

"Hi Bob. I wrote this great new <u>tool</u> that I think will help you with the new thing-a-bob".

A small change yes but <u>tools</u> have the connotation of something that's going to help you do some work. <u>Tests</u> just sound like work. It's been my experience that if you give someone else a test they usually think that they are doing your job. After all aren't you the "tester". But if they have a new "tool" they might be able to do "their" job more easily and will be more likely to try it.

<u>Wow I'm convinced, how do I do it?</u>

The first steps in creating a host test library will be the same as for creating any test library. There are many great seminars on this subject but for now let's assume that you have a plan and want to start creating the tests. In each case you are going to use a display emulator of some kind to open the hosts editor program. Write the code that will generate the desired test application and then, in the case of the AS400, you will compile the code. On the mainframe the code is complied at run time.

One note: There are many variations on this but the host examples I will use are based on an IBM P390 with MVS using pipes with the Fullscreen program. I will use the Rexx program editor. The AS400 I will use to demonstrate from is a V4R7 with the Program Development Manager program as the application editor.

For this example on the Mainframe we will use Rexx programming language and 3270 data protocol. On the AS400 you will use CL
Programming language and use (optional but recommended) the 5250 data protocol. The effort required to learn these languages may seem daunting but after you learn one they tend to build on each other and it begins to get easier.

Let's walk you through your first Mainframe test application program and see just how easy it is. You are going to need to gain access to the host editor. To do so you will need access to a terminal or a 3270 PC emulator. From the emulator log onto your mainframe and from the command line enter the command:

```
xedit testname exec A1 (Enter)
```

This command instructs the host to open the Rexx editor to the file testname, using a file type exec, and to locate the file on drive A.

The edit screen will be generated with input fields where you will enter application code.

Note: /* */ encloses the comments.
Most typical programming operations that you may be familiar with in other languages are available also in Rexx such as loops, conditional expressions, Boolean expressions etc.
Below is a simple sample that will give you an idea of the code structure.

```
/* ***************************************************** */
/* TEST   : REXX BASIC SCREEN EXAMPLE TEST              */
/* PURPOSE: TO SHOW HOW TO DO IT                        */
/*                                                      */
/* DATE :   08/03/00                                    */
/* AUTHOR : BILL ROBSON @ WRQ                           */
/* ***************************************************** */
/* CLEAR OFF THE EXISTING SCREEN                        */

VMFCLEAR

 /* SEND NEW SCREEN DATA COMMAND CODE FOLLOWED BY TEXT    */
 /* DATA EXAMPLE THEN WITH HEX DATA EXAMPLE.              */
 /* COMMAND CODES ARE:                                    */
 /* 80 = ERASE WRITE                                      */
 /* C0 = ERASE WRIRE ALTERNATE                            */
/* 20 = WRITE STRUCTURED FIELD                            */
/* 06 = READ MODIFIED                                     */
/* 02 = READ SCREEN                                       */

/* PLACE THE CONTROL CODE INTO THE OUTPUT VARIABLE OUTPUT*/
/* EXAMPLE OF TEXT DATA                                   */
/* FOLLOWED BY EXAMPLE OF HEX DATA                        */
/* FOLLOWED BY COMMAND THAT SENDS THE DATA                */

   OUTPUT = '80C2'X||,
          'THIS DATA WILL APPEAR ON LINE ONE.'||,
          '11C150D389958540E3A6964B'X
'PIPE VAR OUTPUT | FULLSCREEN | VAR INPUT'

/* EXAMPLE OF READING THE SCREEN                          */
/* THE RETURNED DATA GOES INTO VARIABLE INPUT             */

  OUTPUT = '06'X
'PIPE VAR OUTPUT | FULLSCREEN CONDREAD| VAR INPUT'

/* SEE IF THE EXPECTED DATA WAS RETURNED                  */
IF SUBSTR(INPUT,2,1) = '60'X THEN SAY ‘GOT INVALID DATA’

  SAY EXAMPLE TEST COMPLETE

   EXIT
```

After inserting the data, tab to the command line, type file and (Enter). Your first test Application is saved.
To execute the test you simply type in the name of the test file on the command line and press the enter key. The mainframe will compile on the fly.

Now let's look at the same example using the AS400 with a CL program and the built in AS400 application PDM as the editor. From a terminal or 5250 emulator, logon to your AS400. You could use any existing library but let's create a new test library to store all the applications you will make. From the command line enter the command:
CRTLIB (Enter)
This will take you to the Create Library screen.
On the Create Library Screen enter your new library name, the library type "test" and the library description. (Enter)
Next you will need to create a Physical File within the new Library to save the application code. Do this by typing
CRTPF (Enter)
Enter the new file name and the name of the library you just created (Enter)

You are now ready to put a member (source file) into your new library. To open the editor enter the command:
STRPDM (Enter)

Select the choice "work with members" (3) (Enter)
Put in the name of the new Library and Physical File you just created (Enter)
You will now see a list of current members (empty now) in your new library.
Select F6 to create a new member and the edit screen will appear.

Note: The screens used in AS400 test application can be created using the Screen Design Aid program of the AS400 or by writing all of the 5250 code yourself. Using SDA you need know nothing about 5250 data stream but you lose control the way the screen is created as well. Using your own data to create the screens is more cumbersome and you need to have a full understanding of the protocol but you can then create the application to do almost anything. What you need the test to do will ultimately make the choice for you. The example below is created with user defined code.

The AS400 CL application example looks like this:

```
/****************************************/
/* TEST       :CL EXAMPLE               */
/* PURPOSE: TO SHOW HOW TO DO IT        */
/* DATE       : 08/03/00                */
/* AUTHOR : BILL ROBSON @ WRQ           */
/****************************************/

/* DECLARE A DISPLAY FILE TO CONTAIN THE RECORDS                    */
/* DECLARE A VARIABLE WE'LL USE TO COMPARE RETURNED DATA            */

   DCLF    FILE(BR5250EML/EMLSCR)
         DCL       VAR(&EXPECTED) TYPE(*CHAR) LEN(20) +
                   VALUE('line1 line2')

/* OVERRIDE THE DISPLAY FILE WITH A USER DEFINED ONE                */
/* THIS ALLOWS YOU TO DESIGN THE SCREEN WITH ANY DATA               */

OVRDSPF    FILE(EMLSCR) TOFILE(EMLSCR2)

/* PUT THE DATA FOR SCREEN 1 THE RECORD &FLD001.                    */
/* BYTES 1-5 ARE THE TRANSMIT HEADER CODES                          */
/* BYTES 1-2 ARE THE OUTPUT BUFFER LENGTH.                          */
/* BYTES 2-3 ARE THE INPUT BUFFER LENGTH                            */
/* BYTE 5 IS THE TRANSMIT COMMAND CODE                              */
/* 73X MEANS SEND 5250 DATA WITH PUT AND GET                        */
/* THEN WE SEND THE 5250 COMMANDS, ORDERS AND DATA                  */

/* WE THEN USE THE SNDRCV COMMAND TO SEND THE BUFFER                */
/* AND THEN POLL FOR THE RESPONSE.                                  */

          CHGVAR     VAR(&FLD001) +
      VALUE(X'002100507304400411000 8+
           110101D389958540D695854B+
           110201D389958540E3A696+
           04520008')
          SNDRCVF   RCDFMT(TESTFMT) WAIT(*YES)

/* NOW LETS COMPARE THE RETURNED DATA
          IF        COND(&FLD001 *EQ &EXPECTED) THEN(GOTO +
                    CMDLBL(DONE))
          SNDPGMMSG  MSG('GOT ERROR ON READ')

 DONE:      SNDPGMMSG  MSG('test complete')

          ENDPGM
```

We save the newly entered application with the F3 key and then compile the source using option 14. Your test application is ready to run.

To run the new AS400 test type call “test name” (enter). The first screen you created in the test will be displayed. Data can be entered or read by the test or manipulated in any way that any other application can do it. Your imagination is the only limit.

As you can see from these simple examples the actual coding of the applications is not that hard. With some practice you can create complete multi-page applications that perform any functions in the same way as any your customers host will.

Oops, I fell asleep, what were the key points again?

We are all aware of the advantages of testing in a closed environment. Perhaps by stepping back a bit and looking at the overall picture we can find a way to open at least part of our testing to outside users by placing tests on our host and gain the numerous benefits that it provides.

Once we decide to open the testing up to other users we need to be able to convince them of how doing so will help them. The keys to getting this acceptance from other users are to 1) Simplify the tests to the greatest degree possible. Step away and then simplify them again. And 2) Design the tests to look and work like Tools that will provide a benefit to the user while at the same time generate valid test results.

When the overall concept is accepted some time will be needed to be set aside to get up to speed on the languages / protocols / programming skills that will be required to pull it off. The good news is that the languages / skills needed are all very common and well documented from numerous sources. It can be learned, but some time and effort will be required to make it happen. Time that may be taken away from the testing task at hand. Just keep reminding yourself of the benefits you will see at the end.

Getting a system like I have described in place is not an easy chore and it may not be an easy sell to management or your other departments. The required effort to make it happen may seem daunting at first, but it can be done. And if done well it can bring benefits that are worth the time spent creating them many times over. I started putting this idea in place nearly five years ago and I still get calls from people I have never met, in departments I never see, thanking me for the “tools” they just used to close a customer call.

Mainframe References:

3270 Information Display System
Data Stream Programmers Reference – GA23-0059-07

Rexx/VM Users Guide – SC24-5465-03

Rexx/VM Reference – SC24-5466-04

AS400 References:

Guide to Programming Application and Help Displays – SC41-0011-02

Application Development Tool Set/400
Program Development Manager – SC09-1771-00

Application Development Tool Set/400
Screen Design Aid – SC09-1768-00

CL Programming V4R4 – ST02-4140-00

NOTE: All Listings from IBM Publishing

Contact Information

Bill Robson
WRQ 1100 Dexter Ave N. Seattle WA. 98109
Billr@wrq.com