# Using GUI-based Automated Test Tools to Test Legacy Applications

By

Keith Zambelich

Sr. Software Quality Assurance Analyst

and

Automated Testing Evangelist

## Introduction:

A great many companies today are currently running numerous legacy (character-based or "green screen") applications on a variety of platforms (Mainframe, AS/400, Tandem, Stratus, etc.). Furthermore, there is an increasing trend toward integrating these systems with front-end GUI, Internet, and Intranet applications, and using the Mainframe and AS/400 type computers for back office processing. Some companies are using AS/400s as Internet and Network servers. Most companies are opting to access their Mainframe (etc.) computers via Terminal Emulation from PC Workstations, rather than use "dumb terminals".

Many of these companies desire to automated their testing processes, but find the selection of a test tool that will meet all of their testing requirements difficult, at best. One tool may be better at testing legacy applications, while another is better at testing GUI applications, and still another is better at testing Internet/Intranet applications. What is sought is a tool or suite of tools that can be used to automate testing *across the enterprise*, as the prospect of training Q/A and technical staff in the use of several different tools from different manufacturers is daunting, if not completely unrealistic.

There are several automated test tool manufacturers that provide enterprise-wide testing solutions. The most notable of these include Mercury Interactive, Inc., Segue Software, Inc., Compuware Corporation, and Rational Software Corporation. This article is confined to describing my approach to configuring Mercury Interactive's WinRunner® automated testing tool for optimal use in testing legacy applications, as I am most familiar with this tool. The concepts expressed, however, can be applied to other automated test tools.

## Learning the Screens:

Regardless of the tool and the methodology (record/playback, scripting, etc.) being used, the first thing one must do is to have the tool learn all of the screens in the application under test. Automated Test Tools such as WinRunner®, LiveQuality® (Segue), and SQA Suite® (Rational) were all initially developed as GUI test tools. All such tools need to "learn" each window (screen) in an application and each object within that window. Using Terminal Emulation allows the tool to treat the character-based *screen* as a *window*, and the *fields* within each screen as *objects*. The tool learns the screen and the fields within the screen and places this data in a Screen / Window Definition File. For WinRunner®, this file is called the "*GUI file*".

The problem is that in actuality, character-based screens are *not* windows, and fields are *not* objects. Legacy application screens are designed and defined by using an X-coordinate, a Y-coordinate, and a length value for each field within the screen. GUI-based automated test tools, however, are usually not initially configured to learn character-based screens in this way. Normally, they are configured to identify fields using either "attached text" (text to the immediate left of the field), or index / location value (relative position on the screen). Neither of these configurations work well for character-based screens.

### ☐ Attached Text doth not a Label Make:

Using "attached text" as a means of field identification can create all sorts of problems. First of all, some applications are written such that the labels change, based on certain conditions. If that occurs, the field will no longer be recognized. More importantly, all fields do not have labels. Consider the following example:

Customer Name: ___________________ _ _______________________

What this actually represents is: First Name Middle Initial Last Name.

What is learned is: Name_0 text = "Name" index = 1, Name_1 text = "Name" index = 2, Name_2 text = "Name" index = 3, where "Name" is taken as the "attached text" of *each* field. What happens if the fields are learned while data is being entered?

Customer Name: <u>Joe</u>

Now what is learned is: Name_0 text = "Name", Joe_0 text = "Joe" index = 2, Joe_1 text = "Joe" index = 3. This time, the "attached text" of field_1 is "Name", while fields 2 and 3 have the "attached text" of "Joe". Text, after all, is text. Can you see how this could cause a problem? What happens when the first name changes?

## ☐ Using Index / Location Will Drive You Batty:

All right then, lets not use "attached text" and just use the field's relative position (Index or Location). Now, using the same example:

Customer Name: ____________________ _ _________________________

What is learned is: Name_0 Index = 1, Name_1 Index = 2, Name_2 Index = 3, where each field has a unique value. This looks like it will work. Unfortunately, many character-based screens contain "hidden fields" that only display under certain conditions. Consider this example:

Field_0: _______ Field_1: _______
Field_2: _______ Field_3: _______

When these fields are learned, they have index values of 1 thru 4 respectively. So far, so good. But when the next record is read using this screen, conditions exist such that the following is now displayed:

Field_0: _______ Field_1: _______ Field_A: _______ Field_B: _______
Field_2: _______ Field_3: _______

Fields A and B are now displayed, but were not learned in the original screen. Field_A now has an *index value of 3*, Field_B now has an index value of 4, and Fields 2 and 3 have values of 5 and 6 respectively. The script running is entering data into Field_2 which was *defined* as having an index value of 3 - *but the field that has the index value of 3 is now Field_A*. Guess where the data goes.

## ☐ There Is A Solution:

In character-based applications, fields within screens are developed using an X-coordinate, a Y-coordinate, and a Length. The way to make this work is to configure the tool to learn field attributes as follows:

Obligatory attributes: X, Y Optional attributes: Length

Now let's apply this to the previous example:
Field_0: _______ Field_1: _______
Field_2: _______ Field_3: _______

These fields will be learned as follows:
Field_0: X = 5, Y = 10    Field_1: X = 5, Y = 20    Field_2: X = 6, Y = 10    Field_3: X = 6, Y = 20

(Note that "length" is not learned in this example, as it is not necessary. In fact it would never be necessary unless a field with a different length was displayed at the same location under some certain condition. In that case, *both* fields could be learned, as they can be differentiated by length.)

Now when the "hidden" fields are displayed, the fields that have been learned are still properly identified, and the script will not fail.

Field_0: _______ Field_1: _______ Field_A: _______ Field_B: Hidden data
Field_2: _______ Field_3: _______

Since "attached text" was not used, data displayed in Field_B will ***not*** affect the recognition of Field_2. Data which is supposed to be entered in Field_2 will be entered in Field_2 *no matter what else is displayed on the screen*.

The one disadvantage to this method, is that if a field is added to a screen, or some of the fields are moved down a row, then a number of fields will need to be redefined. All this takes, however, is editing the Screen Definition or "GUI" file and changing some coordinates. You don't need to "re-learn" the screen. Typically this takes about 10 minutes.

The advantages, however, should be obvious, including the fact that by using this method, the screen *does not have to be developed yet*. The definition for the screen can be keyed-in using the field specifications. This can also be used to test whether or not the programmer developed the screen according to the specs or not.

## ☐ Other Screen-Learning Maladies:

Typically, the average character-based application has *hundreds* of screens. This exercise can take *weeks*, so be prepared! Fortunately, most such applications are fairly compartmentalized, allowing you to learn the screens for a particular function, then go on to another function, etc. If you are using record / playback, this allows someone to get busy recording tests using the completed screens for a particular function while the person learning the screens goes on to the next function. A similar approach can also be used if you have opted to write your own automated scripts rather than use record / playback.

Another problem is that the automated test tool has no way to distinguish between actual fields on the screen and "filler" space. The entire screen is learned - fillers and all. This means that you may need to edit the "GUI" file and delete the "filler-fields" that were learned. This does not *have* to be done, but the "GUI" file will be cluttered up with these fields, making maintenance more difficult.
There is another option, especially if the application has many hundreds of screens, and you have a good VB or C programmer on hand. I used this method with a financial application running on an AS/400. It so happens that AS/400 screen files have a particular format, and while countless options can be used, if a rigid standard is being applied to screen development, this becomes a *fixed* format. This allows you to import the AS/400 screen files as "text" (.txt) files and write a program to re-format these into the format required for the Automated Test Tool's Screen Definition File's format. I had a programmer do just that, and we were able to learn scores of screens in *minutes*, without the "fillers" and getting proper field names as extracted from the application file. Writing such a program takes some time, but it can pay off in the long run.

## Test Tool Integration:

A company that is running character-based applications may already be using an automated test tool which operates on the same platform as the application (Mainframe, AS/400, etc.). This company could later decide to use a GUI interface to the legacy application, add Internet/Intranet access, etc. The automated tool they've been using on the Mainframe cannot be used to test the GUI additions, so an additional tool is sought. A tool like WinRunner®, that can be used to test the Mainframe application via Terminal Emulation as well as the new GUI interface. It would probably occur to someone that recreating the thousands of test cases already operational in the character-based tool is going to be a huge effort. The solution is to integrate the tools, so as to make use of the tool that is already in place.

Any automated test tool existing on a Mainframe or AS/400 is going to have screens which the tester uses to operate the tool. Pass/Fail results are usually displayed on these screens. Since the legacy system can be accessed on the PC via Terminal Emulation, so can the test tool. In other words, *you could use a tool like WinRunner® to control the Mainframe test tool, as well as test the GUI interface*.

Consider the following scenario:
1. Data is transmitted to a Mainframe system from various remote locations
2. This data is formatted into records and written to a file
3. A batch job is run which uses this file to populate the Mainframe database with the data AND sends the data to a RISC6000 Server
4. The data received by the RISC6000 is written to an Oracle database on that system
5. A GUI Application running under Windows NT (Client) can access the Oracle database and display, add, or change data
6. Changes to the Oracle database are transmitted back to the Mainframe and the Mainframe database is updated

Let us assume that this company has an automated test tool that can run scripts that start the batch jobs and verify their successful completion. This tool cannot be used to verify the Oracle database or the GUI application, but a tool such as WinRunner® could be used for this purpose, and *could* also be used to automate the same testing process that the Mainframe tool is performing. The most expedient method of automating the testing of the entire process would be to *continue* using the Mainframe test tool to run the Mainframe tests, and develop automated scripts with the GUI-based tool to do the following:

1. Access and operate the Mainframe tool via Terminal Emulation to run the batch program tests
2. Inspect the pass/fail results of the Mainframe tests
3. If the Mainframe tests "pass", run the tests for the GUI application

Scripts could eventually be written to replace the Mainframe tool's scripts, but only if there was a necessity to do so.

Even if the company did **not** already have an automated test tool, it would be *worth exploring* the idea of acquiring **both** a Mainframe test tool and a GUI test tool in order to automate testing across the enterprise. This really depends on the testing requirements, the applications and processes involved, and the tools being considered.