Automatic Testing Report

Joel Chan

July 17, 2002

Contents

1 Introduction

It has been decided to adopt automated testing as part of the testing process in the Language Software Department (LSD). This is part of a bigger initiative in charting a new path in the overall testing procedure with the ultimate aim of automatically generating tests for the FieldWorks (FW) family of applications. Some of the goals include:

- Replacing 95% of manual production of test scripts with script generation
- To begin scripting automated tests

As part of this initiative, QARun and QADirector have been acquired for use in the testing department.

2 Purpose

The purpose of this document is to report on the progress that has been made in automated testing.

3 Difficulties Encountered

There are two main issues that must be balanced for automated testing to be of use to us. Namely, adherence to specs and the ability to interface with the program under test. We must have some way of keeping track of the specs and quickly modifying our tests to reflect the spec changes. At the same time, our test procedure must be robust enough to cope with changes in the implementation.

Other smaller issues that we must address include the maintenance of test scripts and the scope of testing.

Naming of variables is an issue when there are many controls in the program. As the automated testing suite grows, we will have many scripts to maintain.

To aid us in this, a csv file (NameInfoMapTemplate.csv) has been created, listing the main controls in each dialog. This gives us a standard list of variable names to use.

This list also provides known information about each control and includes attach names, ids, control ids, default values, shortcuts, and the next control in tab order. Thus, should the specs change with regard to these details, we will have a simple way of updating our test scripts.

Changes in code are a problem in automated testing. For us, this is especially true when the names of components change. The changes may cause the test to fail when the test tool cannot find the relevant controls.

In an effort to pre-empt such a situation, it was decided to use the id numbers of the components instead of the names. In the source code, a unique string identifies each component. This string is mapped to a numeric id that is used by the program. The string identifiers are most unlikely to change.

These string identifiers have been used in the file above to denote the ids of the controls. Before use, the relevant header files are read and the string identifiers are replaced with the actual numeric ids. The functionality for doing this is provided by the Perl script UpdateIds.plx. This procedure keeps our ids up to date.

Windows have no assigned id and are identified by certain parameters. In the csv file, the attach names are the names in the Object map which QARun uses to identify the windowprs.

4 Problems Encountered Using the Test Tool

There is an inconsistent identification of objects in Data Notebook. Prior to opening the help dialog in Data Notebook via the F1 key, QARun identifies certain objects incorrectly (different from what Spy reports it as). After opening the help dialog, QARuns identification of components agrees with Spys. It was also noted that the id numbers are reported correctly in both cases.

The cause of this problem is unknown. To work around it, a few lines of QARun script were added into the code, causing the help dialog to open immediately after starting Data Notebook. The use of an objects id number seems to circumvent the problem, too. These approaches appear to be satisfactory solutions.

Stability of the tool may pose a problem. Overall, it works well. However, on 2 occasions, both QARun and QADirector refused to start. The problem could not be solved short of uninstalling and reinstalling the programs.

Interference from other programs is an unexplored area. It has been observed that QARun misses some events when the load on the machine is high, e.g. when the anti-virus scan is running. There appeared to be trouble on several overnight tests relating to missed events. It is uncertain if there were any processes scheduled at the time the test was running.

Memory requirements are an area of concern. In most tests, this is not a significant issue. However, when running a long test like the load test mentioned below, the memory used by the test tool increases dramatically. The memory needed exceeded 500 MB after running for about 4 or so hours. Memory needed did not seem to depend significantly on how much data was logged. A possible solution is to use several shorter running tests in place of these lengthy tests.

Portability of tests may make the automatic test scripts less useful. Different platforms have different controls (e.g. the location of the add/remove program) or different requirements (e.g. needing to reboot Win 98 after installation of Fieldworks) that may break some of the tests. Hopefully, the test scripts will be modular enough that few scripts will need to be changed for use on a different platform.

Problems with QADirector. The reports for the tests do not appear when the tests are scheduled for daily execution. The root directory is opened.

5 Types of Scripts Developed

There are a variety of tasks that are often executed in the running of any automated test script. These can be divided into the following categories:

1. Navigation
2. Checks
3. Recovery
4. Testing

Navigation is an important component of any script. It is a very basic function that needs to be robust and simple to use. It must be able to deal with unexpected dialogs that appear, or expected dialogs that are not there.

It was decided to use the idea of encapsulated setup routines to achieve this. For example, if the order of dialogs opening is A - B - C - D and we wanted to setup dialog C. The idea for the routine to setup dialog C is as follows:

1. Check for the presence of C.
   (a) If present and on top. End routine.
   (b) Else close topmost dialog until C is on top.
2. Check for the presence of B
   (a) If B present, carry out action to open C. End routine
   (b) Else, run setup routine for B, carry out action to open C. End routine

This idea is simple, easily scripted and sufficiently robust.

Checks are the means by which we ensure the program is doing what we expect. QARun has commands to carry out basic operations. However, these are too low level to be useful.

Thus, a library of basic functions is being developed. These functions let us execute a variety of operations, ranging from checking for enabled/disabled buttons, to checking for text in controls, to more complex checks like checking the range of a spin controls and the tab order in a dialog.

In the csv file, other information can be included. Currently, definition of radio button groups and spin control groups are supported (made be a edit box and up-down control).

The information contained in this file provides us with the ability to develop a template for basic dialog checks. We can now run through all the controls in a dialog and carry out the correct check on each control type.

Recovery scripts are useful when something unexpected happens. These may be due to runtime errors or asserts. A simple recovery script has been written that checks for assertions and records them in the log if present. It then terminates the application and restarts it. It would be nice to return the system to a known state, say, by restore a backup. However, this is a little complicated and has not been attempted yet.

Testing scripts will be built from the above components.

A breadth first approach has been adopted, carrying out a simple check on each dialog. This will enable us to detect major problems with the program early on in the testing process.

A simple (low IQ) monkey test has been developed that randomly opens dialogs, clicks the mouse, and enters keystrokes. The usefulness of this test has not been determined. A smarter monkey test may be developed at a later time, perhaps with the ability to target specific areas of the system.

Performance tests have been implemented to record certain performance parameters. These include the time for initial starting of the Notebook, restarting it, and importing performance.

A simple load testing script has also been written. It repeatedly creates new entries in the Notebook with randomly generated text.

6 Further Development

Depth testing should certainly be one of the areas we move in when we have gained more experience with using the automated test tool.

Partially automated tests are an area of consideration. Test scripts can be used to provide navigational and input aid to the tester, who is prompted at critical points to provide an evaluation. This allows tests to be replayed accurately should this be one of our aims. It also improves the efficiency of manual tests that cannot be fully automated or are not economical to automate.

Load testing of the database may be useful. Automated scripts can easily create new entries in Data Notebook and can be run overnight. It might give us some indication of performance and allow us to test the program using large databases.

Automating the tutorial may be beneficial. It may be a good idea to run the tutorial repeatedly since many people will be following it on release.

7 Conclusion

A basic automated test setup has been developed. It is hoped that the automated testing of our software will improve productivity and enable us to release a better product.

A Installing and Setting Up QA

Setting up licenses. Execute the following steps after installing the QA component:

1. Start the program.
2. A dialog displays showing the evaluation period left.
3. Select the License button. The License Administration Utility starts.
4. Click the "Add" button and enter 7166@ls-testqaserver into the next 2 dialogs that appear.
5. Exit.

Setting up QADirector database. Execute the following steps after installing QADirector:

1. Map \\LS-TESTQASERVER\DataNoteBookAutoTests\ to drive Q: if not already mapped.
2. Setup an ODBC source to the Access database Q:\DnDirector.mbd
3. Start QADirector's Administrator.
4. Go to File-Select Database. If the Test Execution or Test Management severs are running, shut them down.
5. Select the Access database and the ODBC source set up above.
6. Finish the database selection.
7. Exit.

Setting up QARun database. Execute the following steps after installing QARun:

1. Map \\LS-TESTQASERVER\DataNoteBookAutoTests\ to drive Q: if not already mapped.
2. Start QARun.
3. Click the "Database" button on the login window.
4. Select the Access MDB option and click "Next".
5. Enter Q:\DnAutoTests.mdb as the database and click "Finish".

B Updating Object IDs

Columns in the table. The information described by each column is as follows:

Variable Name The name of the object.

Type The type of the object, e.g. RadioButtonGroup or Edit.

Attach Name / Spin Edit The attach name of a window registered in QARun's Object Map / The spin control's Edit box's variable name

ID Number / Spin UpDn The ID number of an object; insert the string identifier here / The spin control's Up Down control's variable name

Control ID / Spin Low The control ID of the object; use #StringIdentifier / The spin control's lower limit

Short Cut Key / Spin High The short cut key combination to the object; use key combi / The spin control's upper limit

Next in Tab Order / Spin Increment The next object in tab order; use variable name / The spin control's increment

Default value / Spin MaxClicks The object's default value / The maximum number of clicks to try in testing the spin controls

Some columns have multiple meanings. Spin control specifics were included in the list above. For radio button groups, the names of the radio buttons are recorded in the 3rd column and beyond.

Updating the table. Execute the following steps:

1. Save the table as a csv file.
2. Ensure that all header files needed are listed in HeaderFileNames.txt in include order.
3. Run UpdateIds.bat
4. This will run a perl script that reads the header files (from HeaderFileNames.txt) and the template csv (NameInfoMapTemplate.csv). It will then update the ID numbers and output to NameInfoMap.csv.
5. A simple report about the update operation is printed to stdout.