

The Challenges of GUI Testing

Today I'd like to share the experiences of the SAS Institute Quality Assurance community facing the challenge of testing the SAS® System graphical user interface.

The SAS GUI evolved from our classic ASCII and 3270-terminal text interface of the 1980's, our early Display Manager System or DMS. In those days, UNIX and the Macintosh offered the GUI vision of the future. But with snap-together GUI application builders such as VisualBasic and JavaBeans still a decade away, vendors of large applications were obliged to develop their own user interfaces. Additionally, at SAS Institute we have generally preferred to create our own infrastructure rather than rely on outside vendors. For example, when we converted to using C code to develop our products, it also made sense to write and sell our own C compilers. This provided us with crucial control over the reliability and features we needed in the compilers.

For the user interface, this approach meant that we developed a cornucopia of home-grown visual controls, widgets, decorations, and actions. As time went on, these visuals were augmented by new ones provided by each operating system as it became more GUI-oriented – standard pop-ups, dialogs, menu layouts, and so on. More recent complications include the advent of HTML and Web documents, Java, and thin clients. To some degree we have converted to utilizing these native controls, and we have many exciting Web and Java projects under way, but at this stage of the game we have quite a mosaic (if you'll pardon the UNIX pun) of visuals at the basis of the SAS GUI.

Testing a GUI is enormously challenging even when we merely consider the number of test cases we can develop from just one screen in an application. If we multiply all the actions available from the menus, pop-ups and buttons by all the objects they'll act on, it is absolutely amazing how quickly we can wind up having thousands of test scenarios per screen. Granted, by prioritizing the most likely scenarios, and by removing tests that essentially duplicate other tests, we can get good coverage with relatively few test cases; and every tester has been trained to identify important edge conditions to select for the test suite.

Obviously, even if we identify as few as ten to fifty key tests per screen, we still have to multiply this by the number of screens in any SAS application or product. Not only that, but every aspect of the GUI that does not consist of portable SAS widgets may require separate tests on each operating system we support. In Version 7, this means we have to multiply a certain number of the tests by more than ten operating systems. In the testing world there's an adage: "Good, fast, or cheap – pick any two." Obviously, in Quality Assurance we always pick "Good". So we are left trying to balance "fast" with "cheap" as efficiently as possible.

Batch Driver

Clearly, any tool that will capture test scenarios and automate the playback of the test suite will be crucial for testing the system quickly. Up through most of the

Version 6 testing cycles, we used a testing tool called "Batch Driver" for GUI capture and replay. This tool achieved the objectives of being both fast and relatively cheap because it is written and maintained in-house, and the same developers providing GUI controls for the SAS products could help maintain the test tool. The solution worked for us well until we began utilizing more native operating system visuals, which the Batch Driver typically couldn't even "see", and until the GUI grew to be such a large part of the SAS System. So after years of self-sufficiency, we finally were forced by the sheer volume of GUI testing to relinquish "cheap" in favor of "fast", either by devoting enormous resources to our internal tools, or by purchasing GUI testing tools written by outside vendors. In part this was a natural consequence of using more of the native operating system visual controls. As GUIs evolve into modular snap-together objects, with Java applications being a prime example, we would expect to increase our use of the vendor testing tools that will certainly evolve to accompany them. Today I'll describe our use of the internal Batch Driver tool, then highlight its problems and the solutions we have found with GUI testing tools called SQA Suite, WinRunner, SilkTest, and SunTest.

The Batch Driver is implemented as an in-house SAS procedure, or PROC. Since it is fully integrated into the SAS System, it is suited especially for testing portable widgets, and it can be run on any platform and build of the SAS System, whether it is a version optimized for the customer, or a version built with debug features for developers and testers. It has a capture facility, suitable for creating tests by stepping through the test scenario graphically. It renders the steps into an output text file full of fairly arcane statements. The text file is later replayed by an automation tool into background SAS sessions, which typically display on unmonitored terminals. Additional text output is generated during the test, which we compare to benchmarks.

The Batch Driver also has a facility that allows the tester to step through a testcase one batch driver command at a time. This functionality is implemented only on our development platform, currently HP UNIX, but is suitable for isolating failing portions of a test case. The Batch Driver traditionally took a black box testing approach for most SAS products, meaning that the tester would have no knowledge of the state of the graphic objects under test, or the underlying code that manipulates them. Rather, the tester would be concerned strictly with the user-oriented behavior of the GUI. Now, however, the tool has white box functionality for SAS/AF® software. Using a Batch Driver command called "Dumpmode", a tester can capture the state of a SAS/AF object by dumping all or part of the object's attributes to the Batch Driver output file. This allows the tester to provide more thorough information to the developer when isolating bugs.

The Batch Driver is most effective when testing the older SAS GUI products, SAS/FSP® software and SAS/AF software. Older test programs also use the Batch Driver to test SCL functions through the traditional Program Window in the SAS Display Manager System. Because it is an in-house project, Batch Driver tests fit our test library structure very well and are easily managed with our other home-grown tools that run tests twenty-four hours a day. This is not the case with any of the vendor tools we are using. Currently we can use the Batch Driver to exercise about fifty percent of our target GUI features. Given sufficient resources, both in

development (for updating the tool for all platforms), and in the testing community (for designing and maintaining the test scripts), we could potentially cover ninety-five percent of our interactive testing. However, the cost of these resources has grown to the point that the cost of outside vendor testing tools is now comparing quite favorably.

Testing the more recent Version 6 SAS/AF GUI objects has been an uphill battle. This is because the Batch Driver is often developed in tandem with version of the SAS System it is used to test. That means we typically have to manually test new GUI features, since the Batch Driver tool hasn't been updated to cover them yet, so that it's often only after the first time we ship the new features that we can successfully automate the tests. Also, the world of GUI testing is quickly becoming more complicated, with features such as drag and drop functionality and the ever growing array of widgets used in our user interfaces. Updating the Batch Driver to cover these new features in a timely fashion is becoming prohibitively expensive. Updating the tests to utilize new Batch Driver features is also expensive, particularly when these two efforts overlap.

If the Batch Driver addressed all our interactive testing needs, rest assured we probably would have considered packaging it up and offering it for sale for testing your own SAS applications! But there are more specific limitations than the trends I mention previously.

- The first limitation of the Batch Driver is its non-intuitive functionality. There can be a substantial learning curve.
- Another important limitation, which we haven't quantified, is the potential for the Batch Driver to mask a problem in the SAS System, since they both operate with the same underlying support routines. In this regard it has actually been to our advantage to be obliged to test the GUI manually more frequently than we might have wished.
- There are several limitations testing GUI functionality across the different platforms – especially on IBM mainframes. Additionally, the Batch Driver doesn't recognize Pmenus, native host widgets, or drag and drop actions.
- Finally, the Batch Driver requires specific fonts to be used in test capture and replay, to guarantee that the cursor position is consistent across all platforms for the same test case. This means we have to consider manual testing to cover the software appearance with other fonts.

In addition to the limitations of the tool, Batch Driver test cases are growing increasingly difficult to maintain, as many of the new developments in the SAS GUI can render old tests obsolete.

- Simple GUI changes, and any change in widget placement, can require a complete recapture, unless the tester is adept at splicing in portions of recaptured Batch Driver script code.

- Yet the capture language is cryptic. Even though a tester can alter the replay script at its command level, reading the script is very difficult, and mistakes are easily made. The step-through facility used for debugging Batch Driver tests is only available on our main development platform, so that isolating a problem on the other platforms supported by the SAS System requires painstaking effort from the tester to emulate each Batch Driver command up to the point of the error.
- The tester must provide tools to process the captured testware before it is ready to replay.

While the Batch Driver addresses our most critical need, that of testing on multiple platforms, it can't test the newer features of the SAS GUI. I'll turn now to the vendor tools we have evaluated for testing host widgets, advanced SAS GUI features, Web documents and Java interfaces.

SQA Suite Team Test Edition

For testing objects such as Pmenus, as well as for some test coverage in our highly visual SAS/GIS® software and SAS/STAT® software products, one of the tools we utilize is the SQA Suite Team Test Edition from Rational Software. There is an active and very helpful user group community available for assistance with this tool, and our experience with Rational Software's Technical Support has been positive. This package includes an administration system for managing the testware repository. It also offers a system for developing test plans and reporting on tests. There is a tool called SQA SiteCheck that validates all the links in a web site, which we have found especially useful for testing our new browser-based online help system. The engine of the tool is the SQA Robot, which allows us to record and then play back the interactive tests. The overall strategy is to navigate through the application, checking along the way to make sure we are in the right window. Then we verify the windows, objects, and window contents using test cases at certain checkpoints. We expect rapid growth in the use of this particular tool by our testing community. Approximately seventy percent of the Pmenu tests on the Windows platforms can be automated with the SQA Suite, and we believe this will hold true for new test areas such as native Host Editors (which are optional replacements for our DMS Program Editor window).

One of the benefits of this product is the easy startup. After taking an introductory class, as long as the tool recognizes all the objects under test, a tester can easily begin recording the tests. There is very little environmental setup to do with the applications being tested. For example, we don't have to give every object a tag or ID before beginning, as we do with other testing tools. The tool allows us to do image and region captures, which also gives us masks to block out a changing portion of an image – for example, a date. We found it necessary in many cases to introduce additional wait states in order to ensure that the tests would re-play reliably.

One problem we encountered was that the SQA Robot could not recognize the menu icons on the Pmenu items; we have to run the SAS System with a special option to turn these icons off for testing purposes. The SQA Robot still has difficulty recognizing certain SAS widgets. When the widgets are seen but not recognized, there

is a way to "teach" the tool to understand the objects. Unfortunately, this doesn't always succeed. As a work-around for this problem, we are beginning to use the dump mode process as a part of the testing. This will allow us to see all of the attributes of an object. As we teach the SQA Robot to recognize more of our widgets, the tool will become increasingly useful for GUI coverage, if only on Windows platforms. If the product were available on other platforms, it would have enormous potential in our testing community as a broad-based GUI testing tool for SAS applications. Even so, it is a necessary alternative to our Batch Driver when we need to test applications with native host widgets on Windows.

WinRunner

To better meet our cross-platform testing needs, we also use WinRunner and Xrunner from Mercury Interactive, which run on Windows and UNIX, respectively. Our experience with Xrunner is still minimal, but I can say that unlike all the other vendor tools we have used, Xrunner requires a specially instrumented version of the SAS System, since it has to be linked with the SAS System interfaces to Motif. This is non-trivial, and may be contributing to our slower start-up using Xrunner.

So far, only the Enterprise Reporter™ product has been tested with WinRunner. With this tool, the testers developed a fairly sophisticated set of user-defined functions to perform specific Enterprise Reporter tasks that make it somewhat easier to create more complicated tests. They have also developed GUI maps for Enterprise Reporter that can be shared by all testing staff. Our experience testing Enterprise Reporter was that most validation is best done using bit maps. These can become quite large and are subject to change, which forces a re-capture.

WinRunner allowed us to test drag and drop actions without any timing problems. Out of the box, there are GUI checks available to validate objects in a window. We find that the WinRunner capture and replay facility is relatively easy to use, though a specific screen resolution and small fonts were required. An *analog mode* enables capture and replay of SAS/AF widgets and mouse or cursor actions such as drag and drop. It is easy to combine debugging with capture sessions. A C-like programming language called *tsl* can be used to extend the functionality. Testers who are familiar with C can programmatically react to the software under test. Using *tsl*, we can write a test, as opposed to merely capturing a test.

On the other hand, the sophistication of the tool can make the learning curve considerably more challenging than for other tools. And while our hope with the Mercury tools has been to achieve cross-platform tests, we find that the test scripts are not portable between Windows and UNIX. After overcoming the learning curve, though, the tester can at least write similar tests for both operating systems.

SilkTest

For Web testing on Windows platforms, we are using SilkTest, a component of the QA Partner Suite of testing tools from Segue Software. SilkTest is used exclusively for testing CGI based components of SAS/IntrNet™ Software, specifically, the SAS/IntrNet Application Dispatcher and SAS/IntrNet *htmlSQL*.

QA Partner is a PC based tool that works by using the OLE automation properties of Internet Explorer and Netscape. It provides a scripting language for capture and replay of browser events. This language is also used for running single series of testcases and test projects. Using this tool, we have been able to automate about seventy-five percent of the testing of the SAS/IntrNet Application Dispatcher. About the same amount of automation is available for SAS/IntrNet htmSQL.

Silk Test's strength is in testing the user interface (a browser) and accessing web servers on the SAS Institute intranet. By testing on Windows but checking content provided by servers on other platforms, we can test over eighty-five percent of the browser market and one hundred percent of the supported platforms for the SAS/IntrNet component, without having to modify the SAS System. Tests can be captured to be replayed on either the Internet Explorer or the Netscape browsers. This allows us to maintain a single set of testcases and benchmarks for both browsers. The testcases are a mix of HTML text pages produced with the SAS/IntrNet Web Publishing Tools and the Version 7 ODS feature, and graphic images produced with the SAS/GRAPH® GIF and JPEG drivers.

The major limitation of SilkTest is that it is only available on Windows. This leaves us with manual interactive testing on all the other platforms we support. The Java support came later for this tool, and the testing community turned instead to the SunTest JavaStar tool for capture and replay testing of Java Applets and the Java Application components of SAS/IntrNet Software.

SunTest

The SunTest Suite is from Sun Microsystems. We use SunTest for testing SAS/IntrNet Java products. We use the JavaSpec component for unit testing and for Java applications. We will be using the JavaStar component for GUI testing of Java applets, and the JavaScope component as a coverage analyzer to determine the percentage of source code lines we are testing. These tools will allow us to automate about eighty percent of the test area. These are the strengths of JavaStar:

- One set of test scripts work on any Java-compatible platform.
- GUI changes force only minimal testware maintenance.
- We can create reusable building-blocks for faster test writing.
- The test language is familiar to Java programmers, so it is easy to use.

JavaStar does not use pixel mapping for GUI capturing. It knows the name of the component we may be clicking on and captures the component we clicked, not where we clicked. Therefore if all the push-buttons move around, the scripts should still play back without having to recapture.

JavaSpec is a GUI interface which assists testers in writing tests. Everything is written in Java, so if we need more testing code that JavaSpec does not provide, we can easily create a new class. JavaSpec also provides many testing parameters, such as boundary values and valid or invalid arguments to get us started in testing our methods. This tool allows us to write one test, and then JavaSpec will create all

the combinations with multiple testing parameters and different setup environments. Again, these will run on any Java-compatible operating system.

JavaSpec, as shipped, did not have a way of performing file comparisons. When asked, SunTest provided us with a Java class to do this. We wrote extra Java code around the tests to include this feature. Unfortunately, JavaSpec cannot inherit any other test. Therefore all tests must be hard-coded, often repeating the same elements of a class rather than segregating these common elements into an inheritable class. For example, we create benchmarks and compare them in the same way for each test. Currently we must write these features into each test, and if our benchmark process has to change, each test must be updated individually.

Even with this limitation, the potential test coverage on multiple platforms makes the tool an excellent candidate for testing all our current and future Java-based projects.

Conclusion

To summarize, in testing a product as large as the SAS System, we have found it necessary to utilize many tools. We have specialized widgets that were developed internally, we're using new and fast-changing GUI technologies, and we also test on very many operating systems. This forces us to use a mix of the tools available. We use the Batch Driver whenever possible, to achieve our goal of sharing tests across multiple operating systems. When we need to automate tests of newer SAS and native host widgets, we turn to the SQA Suite and WinRunner and Xrunner. For Web content, we use SilkTest, and for Java interfaces, we use SunTest. Depending on your application, you may need only one or two of these tools. The important thing is to know what you need to test, and to try and find a tool to automate GUI testing -- if it is a task you will be repeating often enough to justify the cost of automation.

Contributors to this presentation: Dawn Amos, Stuart Austin, JoAnna Longman, Allen Malone, Sue McGrath, Jim McNealy, and Natalie Morgan.

SAS is a registered trademark or trademark of SAS Institute Inc.
SAS/AF is a registered trademark or trademark of SAS Institute Inc.
SAS/FSP is a registered trademark or trademark of SAS Institute Inc.
SAS/GIS is a registered trademark or trademark of SAS Institute Inc.
SAS/GRAPH is a registered trademark or trademark of SAS Institute Inc.
SAS/IntrNet is a registered trademark or trademark of SAS Institute Inc.
SAS/STAT is a registered trademark or trademark of SAS Institute Inc.
Enterprise Reporter is a registered trademark or trademark of SAS Institute Inc.