Automating tests on a PowerJ web applet with SilkTest

Christopher Lott
Telcordia Technologies, Inc.
445 South Street, MCC 1C-345B
Morristown, NJ 07960
Phone: +1 973 829 2149
21 September 2000

Summary

Testing applications on which SilkTest cannot easily manipulate the GUI objects presents significant challenges. This note presents tips and tricks for testing a web applet built using Sybase’s PowerJ application generator. SilkTest is simply not able to see the fields on these applets. However, the applets use JavaBeans technology, and SilkTest can invoke methods on JavaBeans with ease. This facility, combined with judicious use of computed tags and dynamically instantiated objects, saves the day.

Introduction

The window shown above is part of an applet built using Sybase’s PowerJ 3.0. An experienced SilkTest user, especially someone accustomed to SilkTest’s support for PowerBuilder’s “Datawindow” objects, would probably expect that Silk’s “Record declarations” mode would find lots of objects. At a minimum it should see the menus, the toolbar buttons, and three text-field objects corresponding to the three edit fields, not to mention a few static-text objects for the labels.

Unfortunately, in this applet launched from a web browser and running via Sun’s Java plug-in, SilkTest can only see the menus, the toolbar buttons, and a single “Datawindow” object. No static-text objects! Worse, no text fields! Although SilkTest understands Datawindow objects in native PowerBuilder applications very well, it does not (yet) do

---

1 Copyright © Telcordia Technologies, Inc. All rights reserved.
the same for a Datawindow object that is embedded in a Java applet. Of course it’s possible to click the mouse at absolute pixel positions on the window and to send key presses at the application. While that can drive the application, the problem of reading field values remains, especially for fields that will not accept the input focus.

This was the situation I faced in early 2000. Further, the application was available with both a native PowerBuilder GUI and a Java applet GUI, so a key goal was to develop portable test scripts that could be run on both GUIs. The remainder of this note offers a flexible and robust solution to implementing methods on the Java applet that are the same as those offered by SilkTest for native PowerBuilder applications.

Datawindows in Java Applets

As mentioned, the fields in the example window shown above appear on a PowerBuilder “Datawindow” object according to SilkTest. More precisely, SilkTest can see an object of type “powersoft.datawindow.JDataWindowControl.” When recording the window declarations, SilkTest also lists a large set of methods that it can invoke on that object. The following methods seemed especially relevant:

```plaintext
STRING getItemString (INT i1, SHORT i2);
STRING getItemString_2 (INT i1, STRING str2);
INT setRow (INT i1);
INT setColumn (SHORT i1);
INT setColumn_2 (STRING str1);
```

At this point, it was necessary to learn more about these methods; the names seemed self-explanatory, but I needed more information about the arguments. The applet developers use the same methods that SilkTest can invoke to control the applet’s Datawindow, so I browsed through the Sybase PowerJ 3.0 help. That documentation explained that the arguments of type “INT” correspond to row numbers, arguments of type “SHORT” correspond to column numbers, and arguments of type “STRING” correspond to column names. Apparently the Datawindow maps each field to a column number and name. And of course it can display multiple rows from a query result.

Now that I knew what the methods did, I needed to learn the column information for each field. The developers kindly shared with me their Datawindow definitions, where the field-column correspondences are defined. (This was a deviation from classic “black-box” testing, but unavoidable if test automation was ever going to work.) Once I had the column information, I could use the `getItemString_2()` method to read values from fields, and the `setRow()`, `setColumn_2()`, and SilkTest’s native `TypeKeys()` methods to write values into fields. I chose to use column names because I expected that the column names would change less frequently than numbers.

Finally we were able to write test scripts. We read and wrote fields by invoking methods on the Datawindow object using the appropriate column information. Although the testcases ran very slowly, which seems to be a fact of life when using SilkTest to drive this applet, they worked. However, the test scripts for the web applet version of this application were very different from test scripts for the native PowerBuilder version of
this application. My summer students and I wrote many sets of scripts for each GUI before we saw how to make them identical.

**Dynamic instantiation of Java Datawindow objects**

I was working in parallel on the native PowerBuilder version of this application, on which SilkTest can manipulate Datawindow and “DWEditField” objects easily and extremely quickly. I was frustrated by the effort involved with writing different scripts for each GUI, so I looked for a better way. After some fruitless experimentation, and then inspiration from a presentation given at Segue Software’s Quest’99 conference (by Jeff Hemsley of Autodesk), I hit upon a solution. The solution that is described here makes it possible to provide functionality nearly identical to the support that SilkTest offers for native PowerBuilder Datawindow objects, such as `getRowValue()`.

The key tricks are to declare a class that represents the invisible field objects, and to define methods in that class that use dynamic Datawindow objects. Recall that to manipulate a field on the Datawindow object, a user must invoke a method on the Datawindow object, because that’s where the real field-manipulation methods are defined. Also note that any SilkTest object will reveal its tag via the `WindowTag()` method, and the tag for a child object includes all the required information about its parent. Happily, SilkTest uses plain-text tag strings, in which each component is separated from the next by a slash, so the tag string for the Datawindow object can be built easily from the tag string for the (non-existent) field object on that Datawindow.

The main bit of work is to build a class in support of the (non-existent) field objects. SilkTest allows a user to instantiate objects that do not correspond to any object on a GUI, and what is needed here is a class to support exactly that. This class corresponds to fields on the datawindow object. This class should not derive from AnyWin or other predefined winclass since these objects do not exist as far as Silk knows. The class’s only reason for existence is to define field-access methods. Upon invocation, every method in this class gets the tag for the field object, computes a tag that refers to the parent Datawindow object, and instantiates a dynamic object for the Datawindow object using that tag. Then it can read or write the field using the native methods offered by the Datawindow. An example class is provided in the .t file.

There is one more trick to make things a bit more elegant. What tag is appropriate for a non-existent object? In this case, a column name is just perfect! SilkTest is never going to use the tag string for the field, so the tag can serve as an object constructor argument. Using the tag as a constructor also saves the trouble of defining column information in a global variable or in some other way.

For example, here is a complete tag string that SilkTest builds for the first field (column name “lifetime”) on the example “Password Parameters” window shown at the beginning of this article. The final component, which refers to the field object, is untyped.

```
/[JavaMainWin]Password*Parameters/[powersoft.datawindow.JDataWindowControl$powersoft.datawindow.JDataWindowControl[1]/lifetime
```
Everything up to (but not including) the final forward slash is used to instantiate a dynamic Datawindow object.

And for just one more trick, the constructor can include more than a column identifier. For some fields, it can be helpful to include information about the display format. For lack of a better name, it’s called a format descriptor. This is very useful for fields that have dates and times, since a date-time-type value is not returned the way it’s displayed on the screen. An complete tag string for a date-time field with a format descriptor might look like this:

`/[/JavaMainWin]Password*Parameters[/powersoft.datawindow.JDataWindowControl]$powersoft.datawindow.JDataWindowControl[1]/date+mm/dd/yyyy hh:nn:ss`

The “+” character separates the column name from the field format descriptor; after removing the descriptor, everything up to the final slash is used as the Datawindow tag.

**Conclusion**

The combination of a class for non-existent field objects, dynamic instantiation of Datawindow objects, plus column information in tags yields a solution that is short (in lines of code) and allows for implementation of methods in the base class that work for all objects. We have used this approach to duplicate most of the functionality that SilkTest defines for native PowerBuilder “DWEEditField” objects, but for our use on the Java applet’s Datawindow objects. This enhances the robustness and portability of test scripts. By defining additional methods to hide test scripts from certain details such as the exact method of opening and closing windows, we are able to run most test scripts on both the native PowerBuilder and Java applet versions of the application without modification.

**Configuration**

SilkTest must be configured appropriately before the examples in the .t file will work. The Datawindow control in an applet is normally ignored by SilkTest, unless the option “Show all classes” is on, but that causes other problems. To make the Datawindow object permanently visible, the file Segue\SilkTest\EXTEND\Javaex.ini must include the following line in the [ClassList] section (at the end):

```
powersoft.datawindow.JDataWindowControl=true
```

**Examples**

I’ve supplied a file with an example winclass definition that supports the declaration of (non-existent) Datawindow field objects. The recorded winclass that defines the Datawindow object is named “PowersoftDatawindowJDataWindowControl.” Some methods are defined as virtual so that if they are overridden in derived classes, base-class
methods will invoke the appropriate version. I’ve also included an example of using the
winclass to define the example window shown in the introduction.

An example using the output format descriptor functionality is shown next. A winclass
“JdataWindowControlEditFieldDateTime” (included in the example .t file) defines
appropriate versions of getRowValue() and setRowValue() for date-time data types.

[-] JDataWindowControlEditFieldDateTime LastUpdated
[ ] tag "date+mm/dd/yyyy hh:nn"

Acknowledgements

Many thanks to Durgaprasad Ayyadevara, Mariusz Fecko, Mike Long, and Meera
Murthy for their assistance with implementing and testing this approach.

About the author

Chris Lott is a research scientist in the Applied Research business group of Telcordia
Technologies, Inc., Morristown, New Jersey. He has been working on test automation
and test generation projects for the last three years. Chris earned degrees in computer
science from the Ohio State University and the University of Maryland.

About Telcordia Technologies, Inc.

Telcordia Technologies, Inc., an SAIC company, is one of the world's largest providers
of operations support systems, network software and consulting and engineering services
to the telecommunications industry. The Telcordia software organization, comprised of
Operations Support Systems and Service and Business Management Systems, has been
ISO 9001-certified and has been assessed at Level 5, the highest level of the Capability
Maturity Model®, an industry standard for measuring software development processes
that was developed by the Software Engineering Institute at Carnegie Mellon University.
A leader in the development of Next Generation Network technologies, Telcordia
employs more than 6,500 professionals and has revenues of more than $1.5 billion.
Telcordia (www.telcordia.com) is headquartered in Morristown, New Jersey, US with
offices throughout the United States, Europe, Central and South America and Asia
Pacific.

Trademark Information

Java and JavaBeans are trademarks of Sun Microsystems, Inc.
PowerBuilder and PowerJ are trademarks of Sybase, Inc.
SilkTest is a trademark of Segue Software, Inc.