shortcut

Your Short Cut to Knowledge

Google™ Web Toolkit Solutions

Cool & Useful Stuff

David Geary

PRENTICE HALL
PEARSON EDUCATION

www.prenhallprofessional.com

What This Short Cut Covers 3

Introduction 4

The Example Applications 4

1. Use Remote Procedure Calls to Access Online Web Services 6

2. Incorporate JavaScript and JavaScript Toolkits . . 20

3. Implement Drag and Drop 28

4. Integrate with Hibernate for Database Access . . 63

5. Use Ant to Deploy Your Application in Tomcat . . . 75

6. Use Popups and Deferred Commands 91

Conclusion 111

About the Author 112

What This Short Cut Covers

The Google Web Toolkit (GWT) is a cutting-edge UI framework for Java developers, which lets you create rich, interactive user interfaces using familiar idioms from Java's Abstract Window Toolkit (AWT), Swing, and the Eclipse Foundation's SWT. If you've used any of those frameworks in the past, you're already halfway up the GWT learning curve.

This short cut assumes that you have already installed GWT and have experimented with its basic features. It also assumes that you're comfortable with techniques like implementing event listeners as anonymous inner classes and know how to construct applications using panels and widgets. Some of the more advanced aspects of the GWT are explored in this short cut using two applications: an address book and a Yahoo! trip viewer.

Both applications use remote procedure calls to access information on the server or an online web service. The Yahoo! Trips application also shows how you can incorporate Scriptaculous, a powerful JavaScript toolkit, to apply a useful effect for displaying results. Other cool and useful techniques, including how to implement drag and drop and how to integrate with a database using Hibernate are demonstrated. Since you'll eventually want to move your GWT application to a servlet container such as Tomcat or Resin, the process of deploying a GWT application to Tomcat with Ant is also covered. Lastly, this short cut shows how to use popup panels and deferred commands to provide a much more interactive user interface.

Now that we know how to access server-side calls from the client, let's turn our attention to the client and see how you can incorporate your own JavaScript, or even someone else's, into your user interface.

TIP: Render unto client . . .

The GWT is sometimes criticized for not fully supporting Java on the client. Remember two things, however: first, some Java classes don't make sense in a browser, for example, sockets or files; second, if you do need the full power of Java, you have RPCs to do your heavy lifting.

2. Incorporate JavaScript and JavaScript Toolkits

The GWT exists so that Java developers can easily create Ajax-enabled applications without resorting to JavaScript, and for the most part, you can get by with very little knowledge of JavaScript when you use the GWT to implement your web applications.

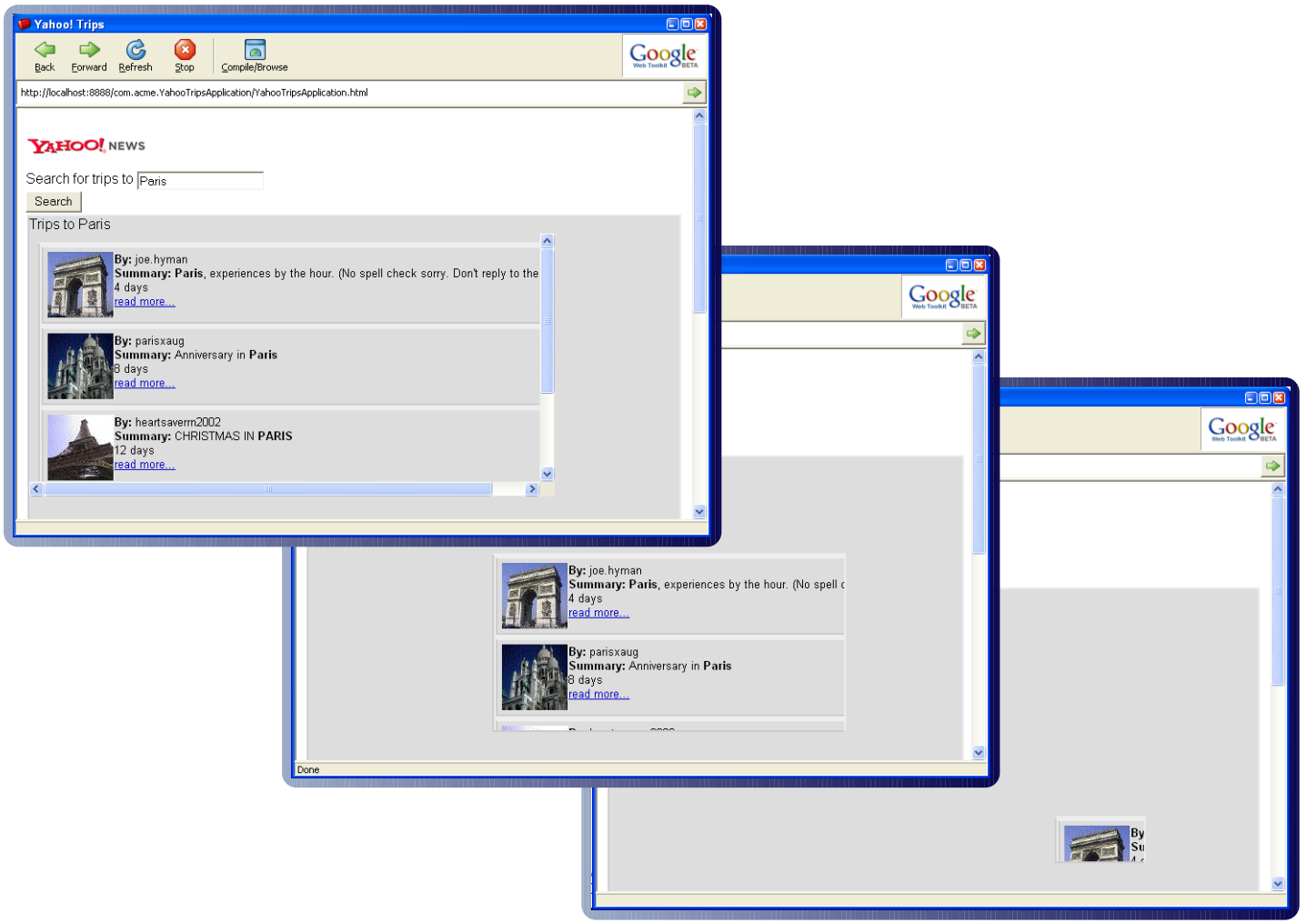
On the other hand, however, there are a number of useful JavaScript toolkits—such as Prototype, Dojo, Scriptaculous, Rico, and so forth—that can greatly increase your productivity and make your applications much more usable. You may also, from time to time, have the urge to dip into some JavaScript yourself.

In the Yahoo! Trips application, we use Scriptaculous to apply a grow effect to a panel that contains the results of our search. The effect is shown in Figure 4.

The GWT makes ingenious use of Java's native methods with something called the JavaScript Native Interface.[3] You declare native methods with commented-out bodies that contain JavaScript. When those native methods are compiled, the GWT incorporates the commented JavaScript. Here's an example of such a native method:

[3] *You may already be familiar with Java's JNI—the Java Native Interface. The GWT's JSNI is a clever take, both in name and in deed, of the JNI.*

FIGURE 4
Using the Scriptaculous grow effect



```
//This is the correct way to implement a JSNI method
private native void applyEffect(String effect, Element element) /*-{
      $wnd.Effect[effect](element);
}-*/;
```

It's vital that you implement JSNI methods exactly as outlined above. The JavaScript can be any valid JavaScript code, but the surrounding native method declaration must follow the preceding syntax. For example, if you change the preceding JSNI method to this (I left off the dashes before and after the braces) . . .

```
//This is an incorrect way to implement a JSNI method: DON'T DO THIS
private native void applyEffect(String effect, Element element) /*{
      $wnd.Effect[effect](element);
}*/;
```

. . . the GWT will not load your application. Instead, when you run the application in hosted mode, the GWT will popup a dialog that says "Failed to load module . . . ." Also, in the Development Shell window you, will be presented with the following ugliness:

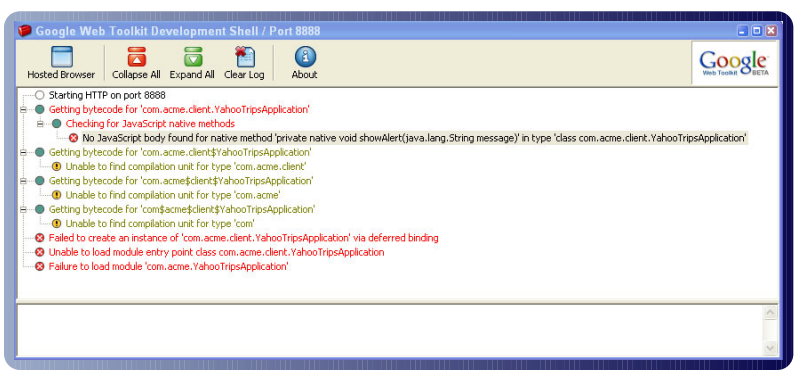


FIGURE 5
The GWT may not load your application if you have a JSNI syntax error

JSNI methods have another special requirement: you must access the window and document JavaScript elements through objects named `$wnd` and `$doc`, respectively.

Incorporate JavaScript Toolkits

JavaScript frameworks such as Prototype and Scriptaculous offer a great deal of functionality that you might like to use in your GWT applications. For example, we could write a JSNI method to perform a Scriptaculous effect:

```
private native void applyEffect(String effect, Element element) /*-{
     $wnd.Effect[effect](element);
  }-*/;
```

Scriptaculous creates an object named `Effect` and adds it to the window object. The preceding code accesses that object to perform an effect.

There are two ways to include JavaScript code in a GWT application. The simplest is to include `script` elements in your application's HTML file, like this:

```
<script src="prototype.js" type="text/javascript"></script>
<script src="scriptaculous.js?load=effects" type="text/javascript"></script>
```

The preceding `script` elements include Prototype and the Effects library from Scriptaculous.

PITFALL: Give <script> elements an empty body

If you use a <script> element to include your JavaScript in your application's HTML file, DON'T DO THIS:

```
<script src="js/prototype.js" type="text/javascript"/>
```

Instead, DO THIS:

```
<script src="js/prototype.js" type="text/javascript"></script>
```

You must have an empty body for the <script> element, or your JavaScript will not be processed. Worse yet, If you don't provide an empty body in the Yahoo! Trips application, the GWT will not call your application's `onModuleLoad` method, meaning the application won't work at all. That can lead to a lengthy and difficult debugging session.

PITFALL: Integrating Scriptaculous on the Mac

In the preceding code fragment, we did this:

```
private native void applyEffect(String effect, Element element) /*-{
    $wnd.Effect[effect](element);
}-*/;
```

That will work fine on the Windows distribution of the GWT, but on the Mac, the same code will result in a nasty exception. The workaround is to do this instead:

```
private native void applyEffect(String effect, Element element) /*-{
    var ne = $wnd._nativeExtensions;
    $wnd._nativeExtensions = false;
    $wnd.Effect[effect](element);
    $wnd._nativeExtensions = ne;
}-*/;
```

The preceding workaround on the Mac is necessary due to an issue with Safari and iframes. In effect, the preceding code forces Prototype to hand-copy the appropriate functions directly to the DOM element the function is passed.

Script Injection for Importing JavaScript

Adding `script` elements to your application's HTML file, as outlined in the preceding section, is one way to include JavaScript code in your application. The GWT provides an alternative mechanism for loading JavaScript files: You specify the JavaScript files that you want to load in your GWT configuration file. For example, Listing 1 shows the Yahoo! Trips application configuration file.

LISTING 1 com.acme.YahooTripsApplication.gwt.xml

```
1.   <module>
2.      <!-- Inherit the core Web Toolkit stuff.                 -->
3.      <inherits name='com.google.gwt.user.User'/>
4.
5.      <!-- Specify the app entry point class.                  -->
6.      <entry-point class='com.acme.client.YahooTripsApplication'/>
7.
8.       <servlet path="/tripService"
9.               class="com.acme.server.TripServiceImpl"/>
10.
11.     <script src="javascript/prototype.js">
12.        <![CDATA[
13.           if ($wnd.$) {
14.              return true;
15.           }
16.           else {
17.              return false;
18.           }
19.        ]]>
```

LISTING 1 Continued

```
20.     </script>
21.
22.     <script src="javascript/effects.js">
23.        <![CDATA[
24.           if ($wnd.Effect) {
25.              return true;
26.           }
27.           else {
28.              return false;
29.           }
30.        ]]>
31.     </script>
32.  </module>
```

The preceding JavaScript checks for the existence of certain JavaScript objects—Prototype's $ function and Scriptaculous's `Effect` object—as evidence that those libraries have loaded. When those objects are detected, the functions return `true` to indicate to the GWT that the libraries have been loaded.

PITFALL: Read this if you can't inject Scriptaculous

This won't work:

```
<module>
   <script src="javascript/prototype.js">
      ...
   </script>
```

```
    <script src="javascript/scriptaculous.js?load=effects">
      ...
    </script>
</module>
```

But this will:

```
<module>
    <script src="javascript/prototype.js">
      ...
    </script>
    <script src="javascript/effects.js">
      ...
    </script>
</module>
```

You must load Scriptaculous files directly, as illustrated by the second code fragment above. Loading `scriptaculous.js` with a request parameter to denote the library that you want to load—which is typically how Scriptaculous libraries are loaded—will not work with the GWT. The technical reasons have to do with a mismatch between the way GWT and Scriptaculous process `script` elements. But regardless of that mismatch, if you always include your Scriptaculous JavaScript files directly, you won't have a problem.

TIP: Import JavaScript in your module's configuration file

Notice that you have a choice of how you import JavaScript. You can either include it with a script element in your HTML page, or you can include it in your GWT configuration file. The former is more familiar, but the latter lets you develop modules that others can use without having to manually include the JavaScript that the module requires.

PITFALL: Go through these JavaScript objects

Don't forget the `$wnd` and `$doc` objects in your JavaScript! For JavaScript that you implement in native methods, you must access the JavaScript `window` object through `$wnd` and the `document` object through `$doc`. If you fail to do so for the `applyEffect()` method from the preceding example, you will get the following exception (look in Development Shell window for that exception):

```
[ERROR] Uncaught exception escaped
com.google.gwt.core.client.JavaScriptException: JavaScript TypeError exception:
'Effect' is undefined
```

(The exception stacktrace is truncated here for brevity)

Now that we've seen how to incorporate JavaScript into your GWT applications, let's look at a totally unrelated topic: implementing drag and drop.

3. Implement Drag and Drop

Drag and drop, which is somewhat of a yawner in the world of desktop applications, is the holy grail for web applications. The ultimate symbol of user interactivity, drag and drop, until recently, has been the purview of mystical JavaScript frameworks such as Scriptaculous. Now, with the advent of the next generation of web application frameworks, such as Rails and the GWT, we're starting to see drag and drop become accessible to the common web developer.

You might assume that to implement drag and drop, we are going to make use of what we learned in "2. Incorporate JavaScript and JavaScript Toolkits" on page 20. But that's not the case. Consider: