Creating dynamic web applications with the Google Web Toolkit

GOOGLE TOOLS

The Google Web Toolkit lets you develop complex web applications in Java and automatically converts them to AJAX apps.

BY RAMON WARTALA

Google is more than just a search engine. This vast and rapidly expanding company is also a major center for software development. Besides programs such as Google Desktop and Google Earth, the company also releases other web-based products once a quarter. While Google Mail is enjoying the limelight, new applications such as Google Reader, Google Calendar, or Google Spreadsheet have attracted very little attention. These lesser known applications share the look and feel of their more popular counterparts, and they use AJAX for quick and easy client access.

Although many suspected Google had its own framework running under the hood, there was no way of knowing for sure until recently. But Google finally confirmed the suspicions at the Java One Fair in May of this year by putting the Google Web Toolkit (GWT) up for grabs as a free download [1].

What's in the Box?

AJAX [2], the asynchronous processing of HTTP requests and responses, along with some help from Javascript and XML, are currently the buzzwords among web developers. AJAX toolkits for programming languages such as Perl [3], Ruby, and PHP are becoming evermore popular. But Google has ventured into new territory with their Java-based framework. Java simply serves as a generator and test language, as AJAX uses Javascript client-side.

But why Java? The main reason is simple bug hunting. GWT gives developers the ability to run and test an AJAX application in what's known as hosted mode. This means running a Java version of the application within a standard Java Virtual Machine. Programmers can use their preferred development environments and debuggers.

After the application is finished, it is compiled into Javascript. The HTML and Javascript code created by this process can be installed on a web server, where it runs in web mode. The component architecture of the GWT framework comprises a special web browser, a widget class library for AJAX-based interfaces, and Javascript implementations of Java standard classes such as *java.lang*, and *java.util*. Besides this, the framework modifies the Javascript code to suit popular web browsers like Mozilla, Firefox, Internet Explorer, Opera, and Safari.

Getting Started

The 22 Mbyte *gwt-linux-1.0.21.tar.gz* package includes documentation and five sample applications, ranging from a trivial *Hello* program, through a widget overview, to a small email application

Listing 1: Postgresql Table for MyAddress

```
01 CREATE TABLE myaddress."names"
02 (
03   id serial NOT NULL,
04   firstname varchar(50) NOT
   NULL,
05   lastname varchar(100) NOT
   NULL,
06   email varchar(128) NOT NULL,
07   CONSTRAINT id PRIMARY KEY
   (id)
08 )
09 WITHOUT OIDS;
10 ALTER TABLE myaddress."names"
   OWNER TO myaddress;
```

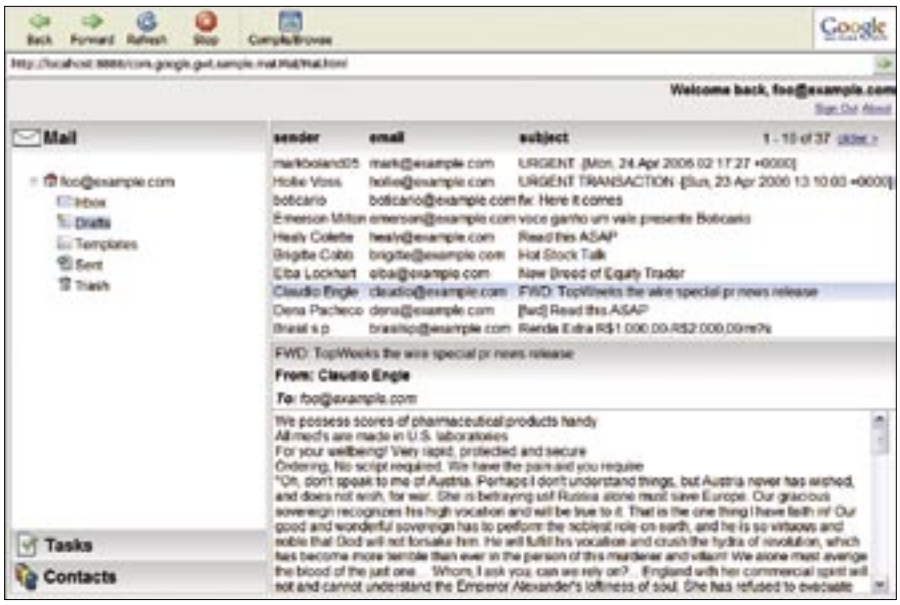

Figure 1: This email program is one of the sample applications intended to demonstrate the capabilities of the Google Web Toolkit.

(see Figure 1). The applications can be launched using the shell scripts in the application directories.

The sample GWT application we will be discussing in this article uses an existing server to query a simple address database. To keep things simple, the server will be based on Ruby On Rails, as the implementation only takes a few lines of code – this has no effect on the client, of course. The finished version can be downloaded at [9]. The MyAddress service developed specially for this purpose is a simple database (see Listing 1) that manages first names, family names, and email addresses.

Listing 2: NameController

```
01 class NameController <
   ApplicationController
02   scaffold :name
03   def find_names_to_json
04     # make sure not to send
   html but text/plain
05     @headers["Content-Type"] =
   "text/plain; charset=utf-8"
06     search_name =
   @params['lastname']
07     names = Name.find(:all, :
   conditions => ['lastname like
   ?', search_name])
08     render_text names.to_json
09   end
10 end
```

The data exchange relies on the JSON format (JavaScript Object Notation) [4]. In contrast to XML, JSON does not use tagging, and thus generates less overhead. To retrieve the address for a family name from the database using Rails, and to package the address in the JSON format, all we need is the 10 lines in Listing 2. Line 6 reads the family name from an HTTP request and finds the matching address in the database in Line 7. Line 8 converts the address to JSON format.

The *ruby script\server start* command calls the MyAddress service. The internal Ruby On Rails developer server gets the service to listen on port 3000 on *localhost*. You could just as easily query the server by entering *http://localhost:3000/name/* in a browser. Another advantage of Ruby On Rails is that you can manage the database via a generated input form (Figure 2). After entering a few records, query them in your browser at the following URL: *http://localhost:3000/name/find_names_to_json?lastname=Name*.

Now let's start developing the GWT project that uses the web service. We can type *projectCreator* on the command line to create the project frame for an application. The *-out* specifies the target directory; *-eclipse* specifies that we will be creating the project for the Eclipse IDE:

```
projectCreator ➥
-eclipse Myaddress_GWT -out ➥
myaddress_gwt
```

The *applicationCreator* command line tool creates the required classes, scripts, and configuration files:

```
applicationCreator ➥
-eclipse MyAddress_GWT -out ➥
myaddress_gwt ➥
de.wartala.client.MyAddress
```

After making sure you have all the required files, you can import the GWT project into Eclipse by selecting *Import | Existing Projects into Workspace* in the Package Manager (Figure 3). Within the project structure, I will be using the XML configuration of a module as the entry point. *applicationCreator* has already created a module configuration with an entry point, based on the required target package (Listing 3). It references the Java class, which the application will call when launched in hosted mode, and is also found in the HTML file, which implements the framework for the client GUI. The most important lines here are the references to the module class and to the GWT Framework's Javascript library:

```
<meta name='gwt:module' ➥
content='de.wartala.MyAddress'>
<script language="javascript" ➥
src="gwt.js"></script>
```

When the application is launched, it first calls the *onModuleLoad()* method, which generates the widgets provided by the GUI library, before instantiating more classes: *MyAddressRequester* in our example. The application then sends requests to the MyAddress service, re-

Figure 2: The web service in our example can provide records directly to the browser.

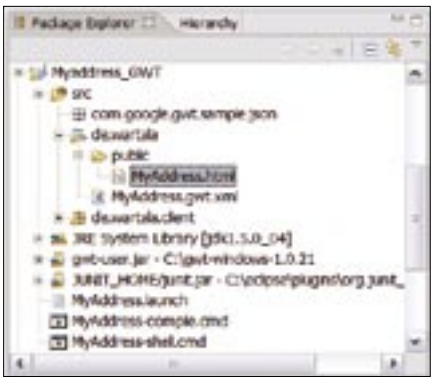

Figure 3: The Google Web Toolkit can optionally create Eclipse project files, giving programmers the ability to import them into the IDE as a project.

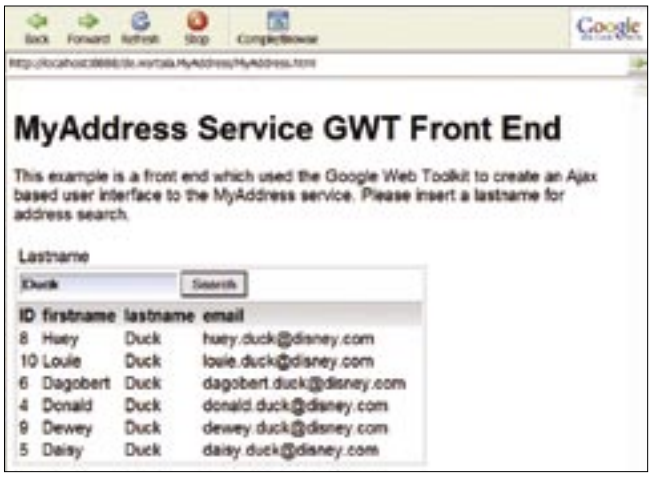

Figure 4: Ajax applications can be debugged using a special web browser in hosted mode.

ceives the responses, and fills the GUI elements with them.

The *initializeMainForm()* method generates the interface, which is comprised of a search button, an input box, and a*FlexTable*. *initializeMainForm()* then sets attributes and events, just like an AWT or the Swing interface.

Our example only requires a single *ClickEvent* to trigger a click on *Search*. The response for this event is implemented by the inner class, *SearchButtonClickListener*.

An *onClick()* event triggers the AJAX part of the application and sends an asynchronous HTTP request to the service, which passes the result to a matching response handler. *JSONResponseTextHandler*, another inner class, implements the *onCompletion()* method, which is called when the asynchronous HTTP request returns any results.

As the service returns a JSON object, we first need to decode the object and break it down into its component parts. *JSONParser.parse (responseText)* handles the task of decoding the object, and the method *displayJSONObject()* handles the latter step, delegating the chore to the method *updateAddressTable()*. The method *updateAddressTable()* renders the results as a table, entering the values from the JSON response in the corresponding rows and columns.

Now for the *MyAddress-shell.sh* command line script. Figure 4 shows the front-end in hosted mode. After entering a family name, the data returned by the service appears in the table.

Hunting Bugs

The advantage of hosted mode becomes apparent if a program error occurs; it is easier to find a bug in the Java code than in the compiled Javascript. Setting the *-eclipse* parameter when calling *projectCreator* creates a file with a *.launch* suffix besides the project-specific data. Thanks to the parameters configured here, the application can be executed in Eclipse and debugged with a little help from breakpoints and other techniques (Figure 5).

Listing 3: Client Entry Point Class

```
01 import com.google.gwt.core.client.EntryPoint;
02 import com.google.gwt.user.client.ui.RootPanel;
03 import com.google.gwt.user.client.ui.TabPanel;
04
05 /**
06  * Entry point classes define <code>onModuleLoad()</code>.
07  */
08 public class MyAddress implements EntryPoint {
09
10   /**
11    * This is the entry point method.
12    */
13   public void onModuleLoad() {
14     TabPanel tp = new TabPanel();
15     MyAddressRequester myJson = new MyAddressRequester();
16     tp.add(myJson.initializeMainForm()  ,"Lastname");
17     tp.selectTab(0);
18     RootPanel.get().add(tp);
19   }
20 }
```

INFO

[1] Google Web Toolkit: *http://code.google.com/webtoolkit*

[2] AJAX: *http://en.wikipedia.org/wiki/Ajax_%28programming%29*

[3] AJAX and Perl: *http://www.linux-magazine.com/issue/62/Perl_AJAX.pdf*

[4] JSON: *http://www.json.org*

[5] GWT Widget Gallery: *http://code.google.com/webtoolkit/documentation/com.google.gwt.doc.DeveloperGuide.UserInterface.WidgetGallery.html*

[6] GWT Widget Library: *http://gwt-widget.sourceforge.net*

[7] gwtPowered.org: *http://gwtpowered.org*

[8] GWT group on Google Groups: *http://groups.google.com/group/Google-Web-Toolkit*

[9] Sample server and client from this article: *http://www.linux-magazine.com/Magazine/Downloads/74/gwt*

In addition to supporting Eclipse-based debugging, the GWT framework also supports unit tests of its own classes. The *GWTTestCase* class is the entry point that implements JUnit integration. The *junitCreator* command line tool generates all the required files, including the test class proper:

```
junitCreator.cmd -junit ➥
eclipse/plugins/org.junit_3.8.1➥
/junit.jar -eclipse ➥
myaddress_gwt2 -out ➥
myaddress_gwt2 ➥
de.wartala.myaddress.test.➥
MyAddressTest
```

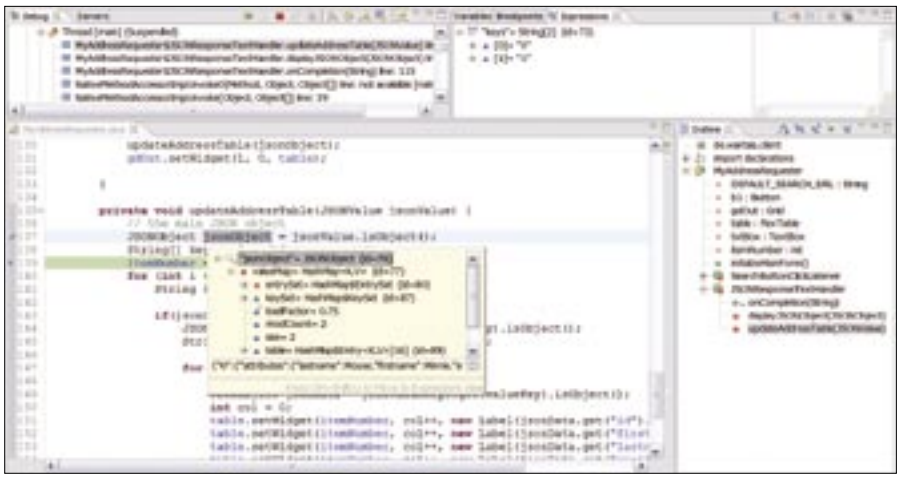

Figure 5: Eclipse Debug Mode speeds up bug hunting.

The files created here are used for test purposes in both hosted and web mode, both in Eclipse and on the command line. If the application runs without an error, you can run the *Project-compile.sh* script to create a Javascript version of the Java application.

The script performs the tasks of copying and generating all required files to and in a *www* subfolder of the working directory when you call *junitCreator.cmd*.

GWT and More

Of course, the Google Web Toolkit has more potential than a simple example can demonstrate. It has a total of 20 widgets; the documentation gives you an overview in the Widget Gallery [5]. Besides HTML equivalents of checkboxes and radio buttons, Layout Managers (Panels) are particularly useful. *VerticalPanel* can help you aligning buttons vertically, for example.

GWT also gives developers the ability to design their own widgets; surf to [6] and [7] for examples and howtos. The framework has already attracted a lively community that uses Google Groups [8] to discuss questions on the subject. ■