Creating GWT Applications in Eclipse

By Patrick Canny

Abstract

This paper describes how to create a Google Web Toolkit ("GWT") application in Eclipse v. 3.5, a.k.a. Galileo, which implements Runnable User Interface ("UI") web components. The paper begins with the set-up process, or tutorial, for a GWT application utilizing the GWT Plug-in for Eclipse. The paper then goes on to explain the methodology for creating Runnable Widgets to be added to the new GWT application.

1 The Problem

Google Web Toolkit is an open-source set of web application tools created and maintained by Google Inc. ("Google") of Mountain View, California. GWT 1.0 was released on May 17, 2006. The current version, GWT 2.0, was released on December 8, 2009. GWT enables web developers to create and maintain complex JavaScript front-end applications in Java. GWT is used primarily for common Ajax problems, including asynchronous remote procedure calls and cross-browser portability.

GWT applications include Widgets, which are User Interface ("UI") components that are synthesized into Hyper Text Markup Language ("HTML") for display in a web browser. These Widgets are very similar to common Java GUI components in the Swing class. They include Button, PasswordTextBox, Radio Button, List Box, Check Box, etc. The developer adds Widgets either directly to the RootPanel (main panel) or to "sub panels", which are then added to the RootPanel for display on a web page. A gallery of GWT Widgets can be found at http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=DevGuideWidgetGallery

Eclipse.org has created a Plug-in for their Eclipse Integrated Development Environment ("IDE") that enables developers to easily create GWT applications using Eclipse. This paper will first focus on the utilization of this Eclipse Plug-in in creating GWT applications. Documentation is available online for installing Eclipse and the Plug-In, thus the focus of this paper will be how the user creates a basic GWT application after setting up the IDE. The second portion of the paper will focus on modifying existing source code for several GWT Widgets in order to implement the Runnable interface.

2 The Google Plugin for Eclipse

Google has partnered with Eclipse.org to create a Plug-in ("Google Plugin") for the Eclipse IDE that, according to Google, is "the fastest way to start developing Google Web Toolkit and App Engine Applications." After downloading and installing the latest version of Eclipse (Eclipse 3.5 is used for this paper) from http://www.eclipse.org , the user then follows the set-up instructions for the Google Plugin for Eclipse, which can be found at http://code.google.com/eclipse/docs/getting_started.html

After setting up the Google Plugin for Eclipse, the user can then create a basic GWT application by creating a new GWT Project in Eclipse. To do this, the user needs to select:

File -> New -> Web Application Project from the Eclipse menu

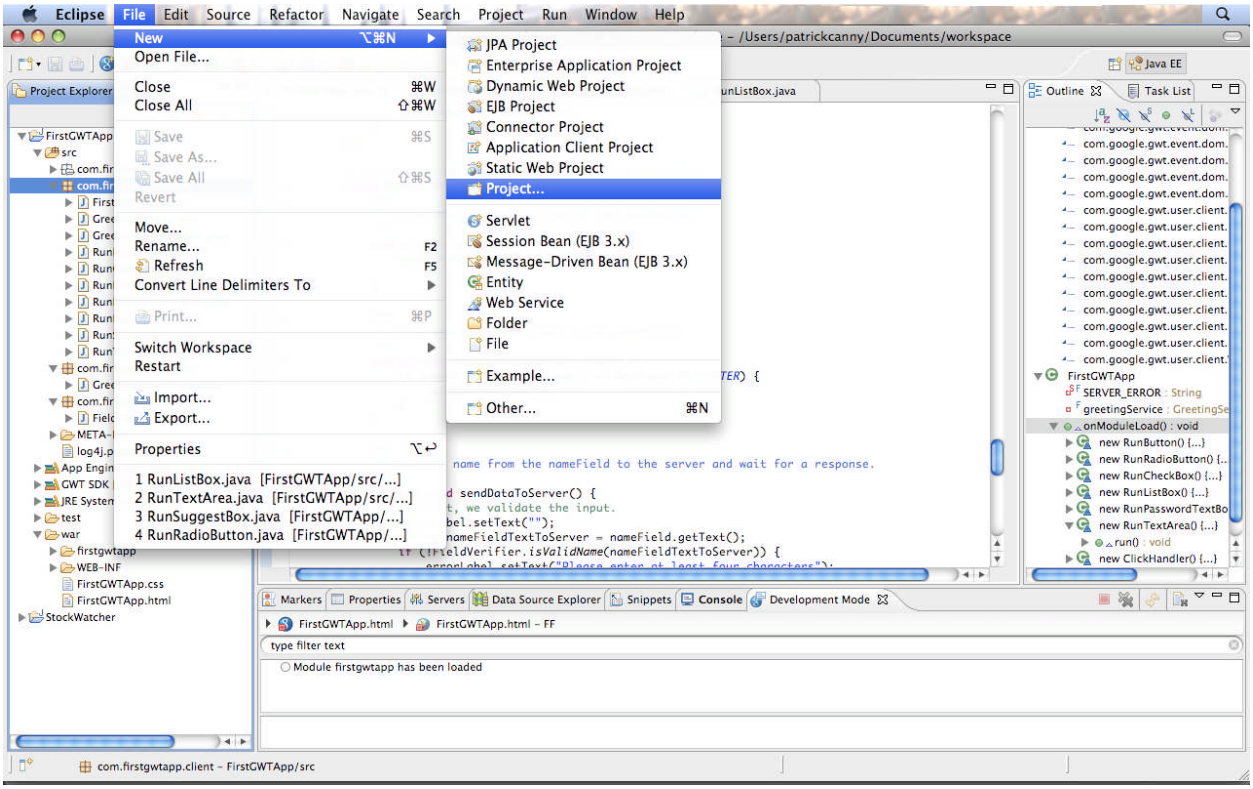

Figure 1 : Selecting a New Project

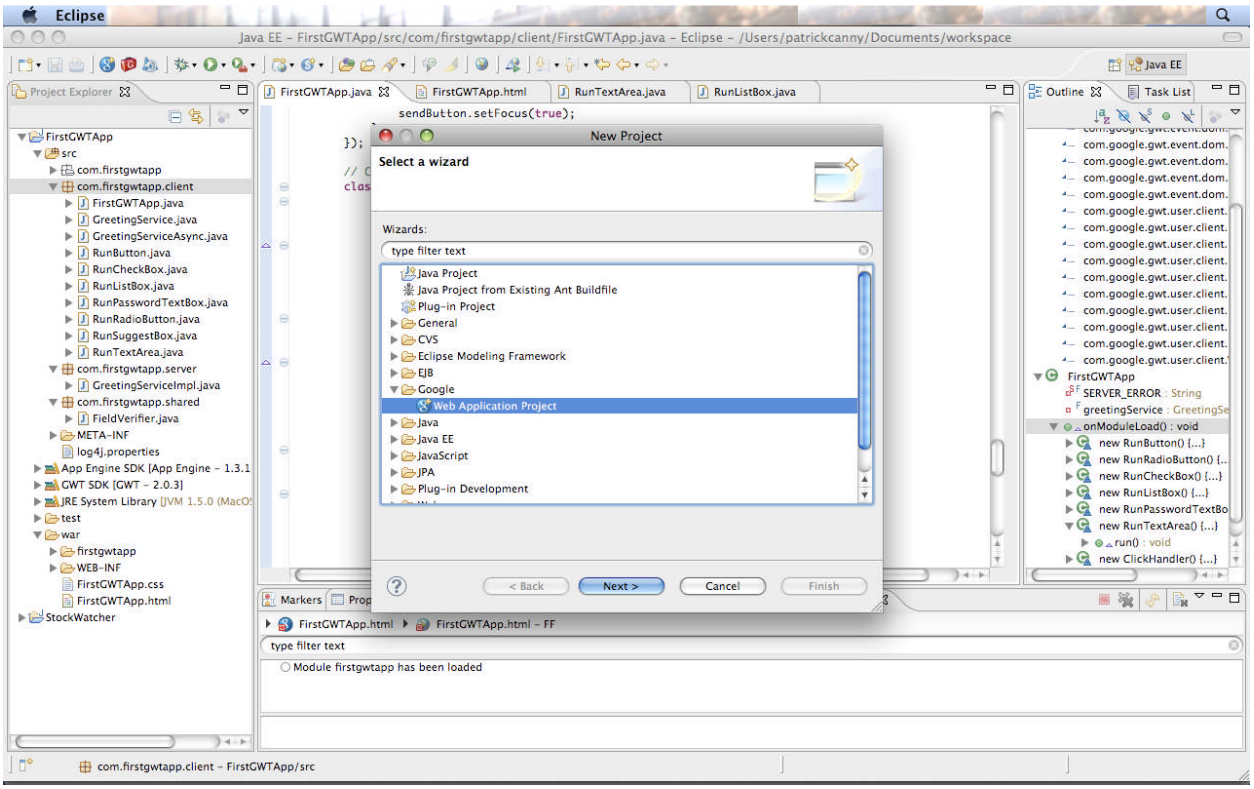

Figure 2 : Selecting a Google Web Application Project

In the New Web Application Project wizard, the user then enters a name for the project and a Java package name, e.g. com.firstgwtapp, then Finish.

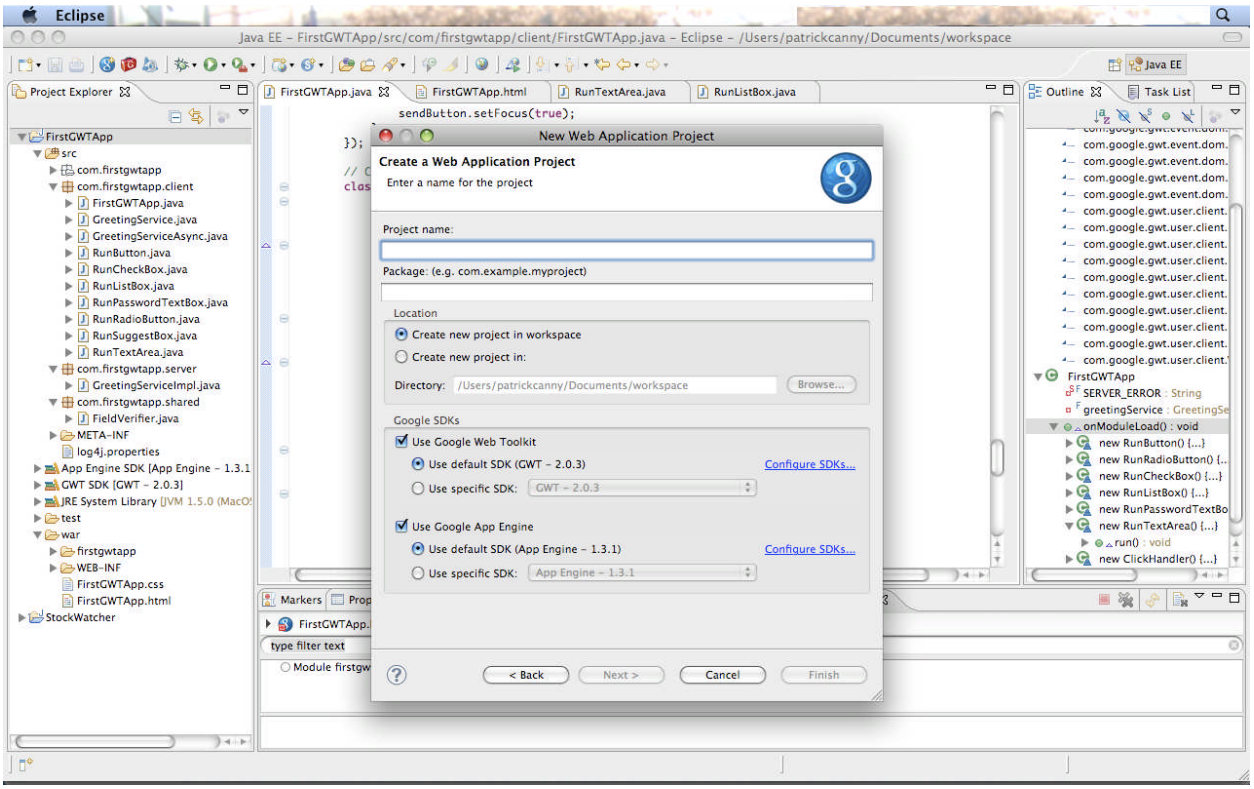

Figure 3 : Entering Project Name and Package Information

This will create an App Engine and GWT-enabled web application.

3 The Parts of a GWT Project

Each GWT module contains at least the following four parts: the Module XML File, the Host Page, the Application Style Sheet, and the Java Source Code.

3.1 Module XML File

The module's XML, Extensible Markup Language, file contains the definition of the GWT module. This is the collection of resources that comprise the GWT application. Within the module's XML file, the developer specifies the application's entry point class. For this module, the entry point class is "com.firstgwtapp.client.FirstGWTApp.java". This can be thought of as the "main" class for the module, but the developer can choose any class within the module as the entry point class.

The source code for the Module XML File can be found in Appendix B.

3.2 Host Page

The module's Host Page is an HTML document in which the GWT web application's code is executed. For this module, the Host Page is "FirstGWTApp.html."

The Host Page references two key components of the module: the Cascading Style Sheets ("CSS") style sheet, "FirstGWTApp.css", and the path of JavaScript source code generated by GWT responsible for the dynamic elements on the Host Page.

The source code for the Host Page can be found in Appendix C.

3.3 Application Style Sheet

The Application Style Sheet is a CSS file containing style rules for the application written in a specialized markup language called CSS. These style rules govern the presentation of a web page's contents, including layout, colors, and fonts. The Application Style Sheet for this module is "FirstGWTApp.css"

The source code for the Application Style Sheet can be found in Appendix D.

3.4 Java Source Code

The Java Source Code is a collection of Java classes that determines the functionality of the GWT application. The source code is broken up into three packages: "client", "server", and "shared."

The package names correspond to the functional location of the corresponding classes within the GWT application. Client side classes are located within "client"; server side classes are located within "server"; shared classes are located within "shared." The majority of the functionality of this GWT application revolves around the client side; however there are classes within the server and shared packages.

For this module, the client side Java classes define the functionality of several Widgets or dynamic UI elements of the Host Page. Each Widget is defined in its own Java class within the module. The next section will describe the creation of these classes.

The Java Source Code for this module can be found in its entirety in Appendix A.

4 Implementing Runnable Widgets

The purpose of this paper is to demonstrate the implementation of the Runnable interface within GWT Widgets. This was accomplished by creating new Java classes that extended existing GWT-enabled Java classes for GWT Widgets, implementing the Runnable interface, and adding these new Widgets as dynamic elements of the page.

4.1 The GWT-enabled Parent Classes

The following GWT-enabled Java classes were chosen to be used in this module:

Button
CheckBox
ListBox
PasswordTextBox
RadioButton
TextArea

All of the above classes are located within the “com.google.gwt.user.client.ui” package from the GWT source code.

In addition, the implementation of a Runnable SuggestBox was also considered. However, upon trial implementation, the functionality proved undesirable and tedious to design correctly. Therefore, the six classes above were chosen for actual implementation.

4.2 The Runnable Classes

Each GWT-enabled Java class listed above was used in the creation of new Java classes that implemented the Runnable interface. This resulted in the creation of the following classes:

RunButton
RunCheckBox
RunListBox
RunPasswordTextBox
RunRadioButton
RunTextArea

The creation of each of the Runnable classes listed above involved the following general process:

- The class is made abstract
- The parent class is imported from “com.google.gwt.user.client.ui”

    - e.g. RunButton imports com.google.gwt.user.client.ui.Button
- The parent class' methods are extended
    - e.g. RunButton extends Button
- An appropriate subclass from "com.google.gwt.event.shared.EventHandler" is chosen depending on the nature of the widget.
    - e.g. RunButton and RunCheckBox implement ClickHandler to handle mouse clicks on the Widget
- The Runnable interface is implemented
- Within the Constructor, the current instantiation is added as an appropriate EventHandler
    - e.g. Within the RunButton constructor: "addClickHandler(this)"
- The inherited EventHandler method is overridden as "run()"
    - e.g. Within RunButton, the method "onClick" is overridden as such:

```
public void onClick(ClickEvent event) {
        run();

} // end onClick
```

4.3 Adding Runnable Widgets to the Application

The next step is to add the Runnable Widgets to the application. This is accomplished by adding the Widgets to the RootPanel, which is the master panel for the web page. Each Widget is added by instantiating a Runnable Widget and overriding the run method within the instantiation itself.

In non-GWT uses of Runnable GUI components, the actionPerformed method is mapped to the run method. In most cases consulted in the creation of this paper, the actionPerformed was simply to output a String to the console. This is not possible in this case, therefore an alternative action is performed. This action is the instantiation and display of a popupPanel. When the user performs a given action on a Widget on the Host Page, e.g. the RunButton, a panel pops up with text indicating to the reader that they performed the action on a Runnable Widget.

The following is an example of how a Runnable Widget was added to the page:

```
        RootPanel.get("runRadioButtonContainer").add(new
RunRadioButton("RunRadioButton","Runnable!",true){
            public void run(){
                final PopupPanel popupPanel = new PopupPanel(true);
        popupPanel.setPopupPositionAndShow(new PopupPanel.PositionCallback() {
           public void setPosition(int offsetWidth, int offsetHeight) {
             int left = (Window.getClientWidth() - offsetWidth) / 3;
             int top = (Window.getClientHeight() - offsetHeight) / 3;
             popupPanel.setPopupPosition(left, top);
           }
         });
        popupPanel.setWidget(new Label("This is a pop-up from a RunRadioButton"));
```

```
popupPanel.show();

} // end run method

    } // ends instantiation of RunRadioButton


    ); // add RunRadioButtonContainer to RootPanel
```

A nearly identical approach is taken for each Runnable Widget.

In order to make sure each Runnable Widget is displayed on the Host Page, the HTML for the Host Page was updated such that each Container is placed within a table on the page. The following HTML source code is used to create this table:

```
<table align="center">
    <tr>
      <td colspan="2" style="font-weight:bold;">Please enter your name:</td>
    </tr>
    <tr>
      <td id="nameFieldContainer"></td>
      <td id="sendButtonContainer"></td>
      </tr>
    <tr>
      <td id="runButtonContainer"></td>
    </tr>
      <tr>
      <td id="runRadioButtonContainer"></td>
      <td id="runCheckBoxContainer"></td>
      </tr>
      <tr>
      <td id="runListBoxContainer"></td>
      </tr>
      <tr>
      <td id="runPasswordTextBoxContainer"></td>
      </tr>
      <tr>
      <td id="runTextAreaContainer"></td>

    </tr>
    <tr>
      <td colspan="2" style="color:red;" id="errorLabelContainer"></td>
    </tr>
  </table>
```

The “nameField” and “sendButton” Containers are used to demonstrate Remote Procedure Calls, and will be discussed further in Section 5.

The application is then run within Eclipse. Within the Development Mode window, a URL is provided to the developer to test the application’s functionality within a browser.

By copying and pasting the URL into a given browser, Mozilla Firefox was used for this paper, the developer can view the application as an interactive web page.

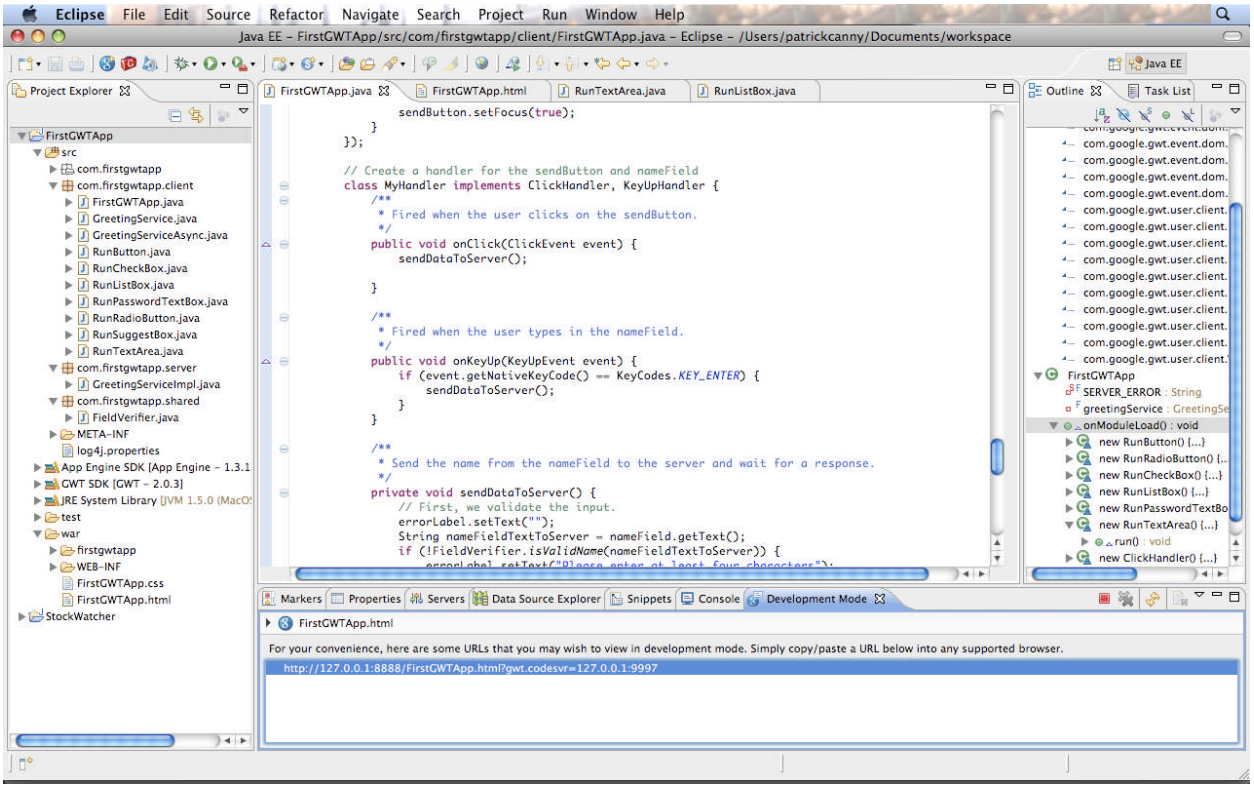

Figure 4 : URL Generated in Development Mode in Eclipse

The following table describes the actions that are performed for given Events on Widgets:

Widget	Event	Text Displayed on PopupPanel
RunButton	Click	This is a pop-up from a RunButton
RunCheckBox	Click	This is a pop-up from a RunCheckBox
RunListBox	Change	This is a pop-up from a RunListBox
RunPasswordTextBox	Key Up	You're typing in a RunPasswordTextBox!
RunRadioButton	Click	This is a pop-up from a RunRadioButton
RunTextArea	Key Down	You're typing in a RunTextArea!

Table 1 : Actions Performed For Events Detected on Widgets

Figures 5 and 6 below show an example of a PopupPanel being displayed after an Event is detected on a Widget.

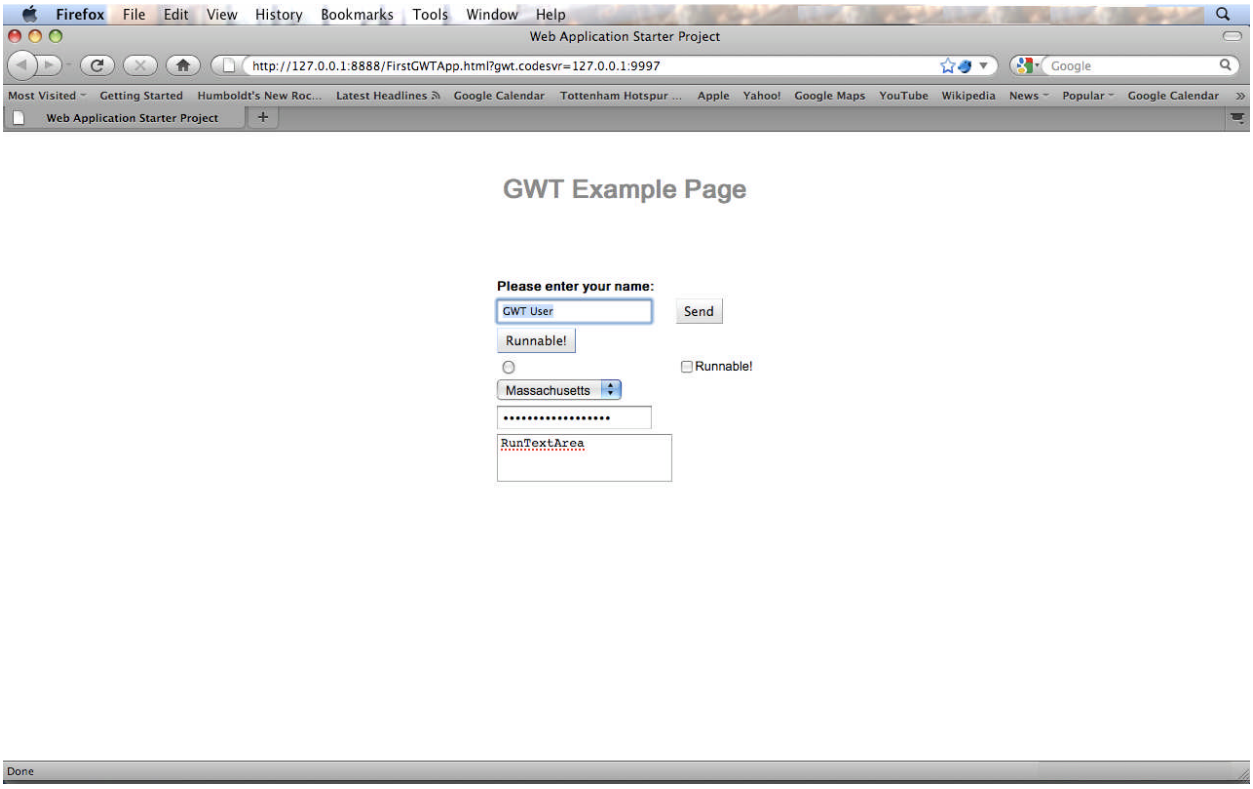

Figure 5 : GWT Example Page on Module Load

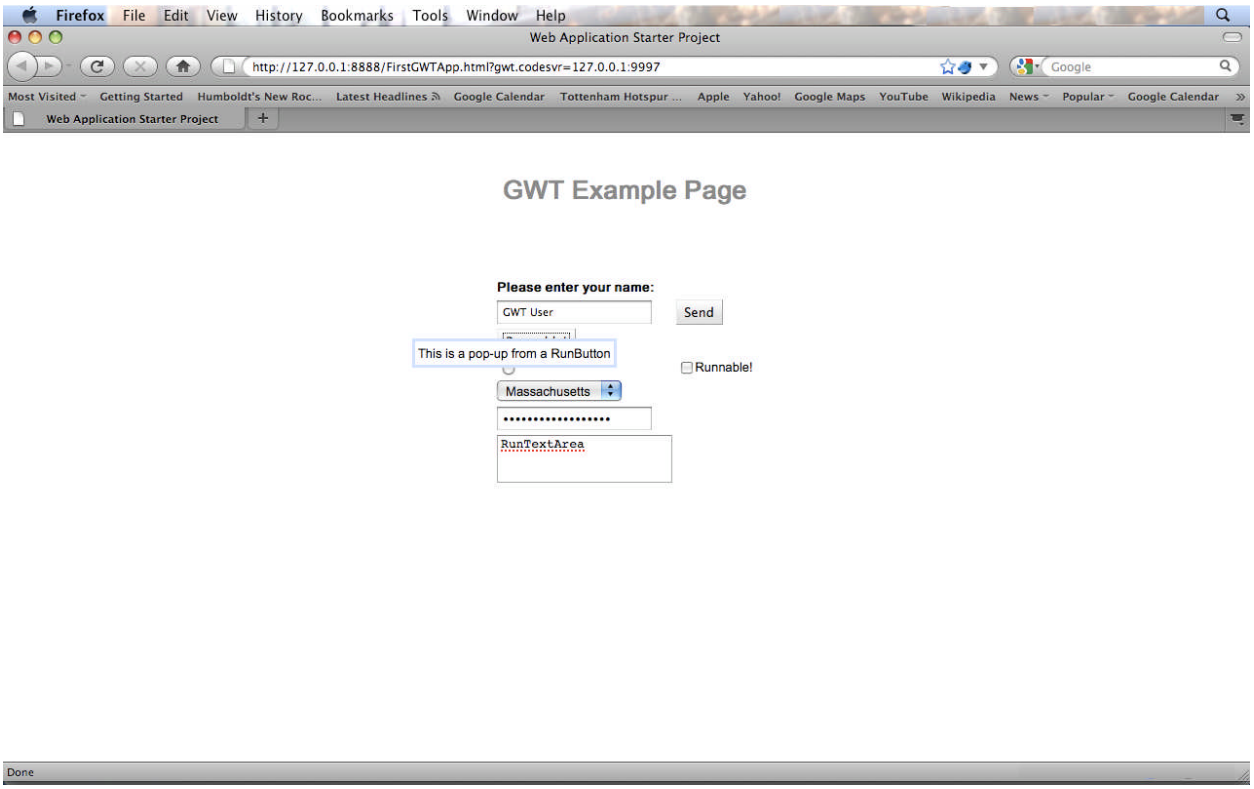

Figure 6 : GWT Example Page - RunButton Clicked

5 Miscellaneous Functionality

In order to demonstrate the capability of GWT applications as client and server side applications, an example Remote Procedure Call was created. This was done entirely within FirstGWTApp.java. This example was created from a Tutorial found online. Since this was done outside of the Problem scope, this section will simply describe the code already present in FirstGWTApp.java in order for the reader to understand the additional functionality present on the Host Page.

A TextBox called “nameField” and a Button named “sendButton” were created and added to the page as “nameFieldContainer” and “sendButtonContainer,” respectively. In order to perform user input error detection, a Label named “errorLabel” was also created and added to the page. The user enters their name into the TextBox, then clicks the Send button to send the name to the server.

Within the FirstGWTApp class, a method called MyHandler was created to handle the actions performed on the Widgets described above. The method sendDataToServer is called within MyHandler.

The method sendDataToServer first checks for user error in entering their name. The only requirement for a correct user name is that it be at least four characters. If the user enters fewer than four characters, the errorLabel is displayed indicating to the user:

“Please enter at least four characters”

The errorLabel is displayed in red text on the Host Page.

The FieldVerifier.java class, located within the “shared” package, is used to check the user name for validity. The isValidName method performs this by checking the length of the name String. If the length is greater than 3, a Boolean TRUE is returned. Otherwise, a FALSE is returned.

The GreetingServiceImpl.java class is the server side implementation of the Remote Procedure Call service. It extends the RemoteServiceServlet.java class and implements the GreetingService interface. The greetServer method throws an IllegalArgumentException. If the name is not valid, according to FieldVerifier.isValidName, the Exception “Name must be at least 4 characters long” is thrown, which becomes the errorLabel displayed on the Host Page to the user to indicate they need to try again.

If the user name is valid, a String with “Hello”, the user name, the server information, and user agent information (Operating System), is sent back to the client side for display in a DialogBox. This functionality is shown in Figures 7 and 8 below.

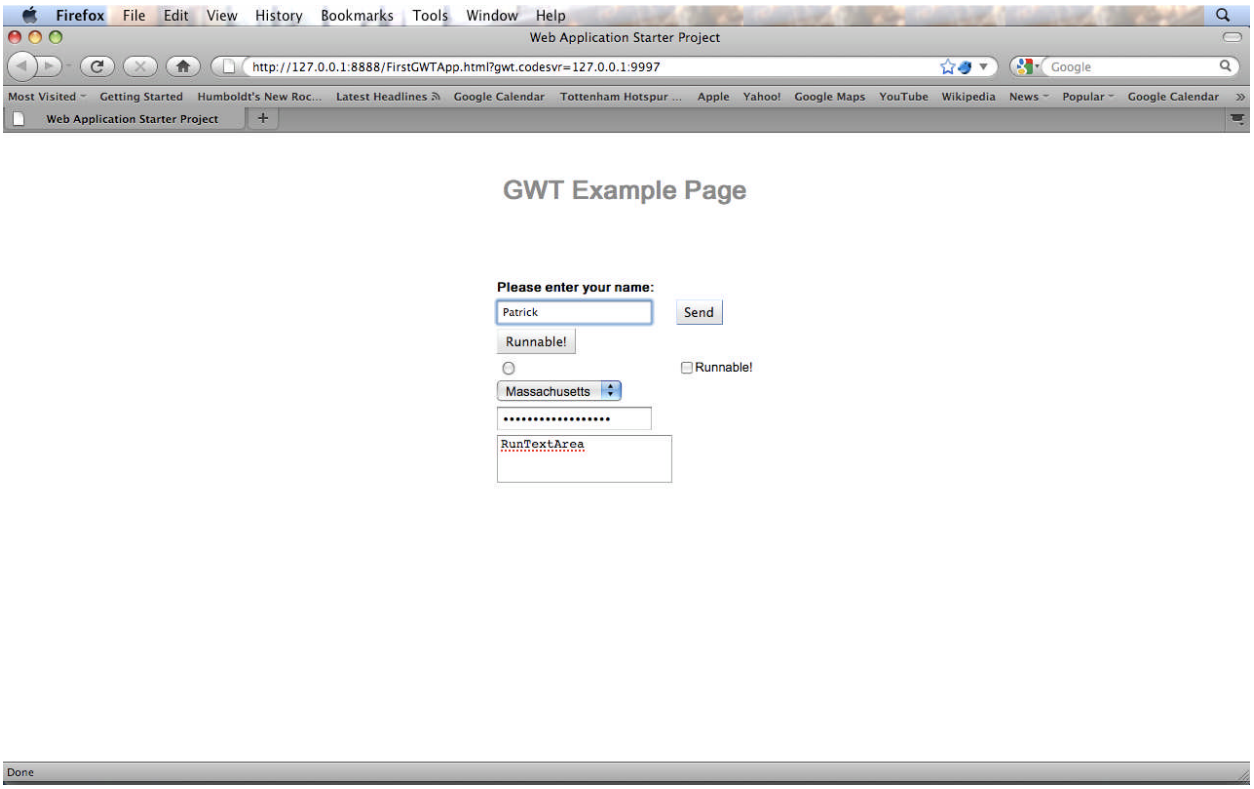

Figure 7 : User Enters Name into TextBox

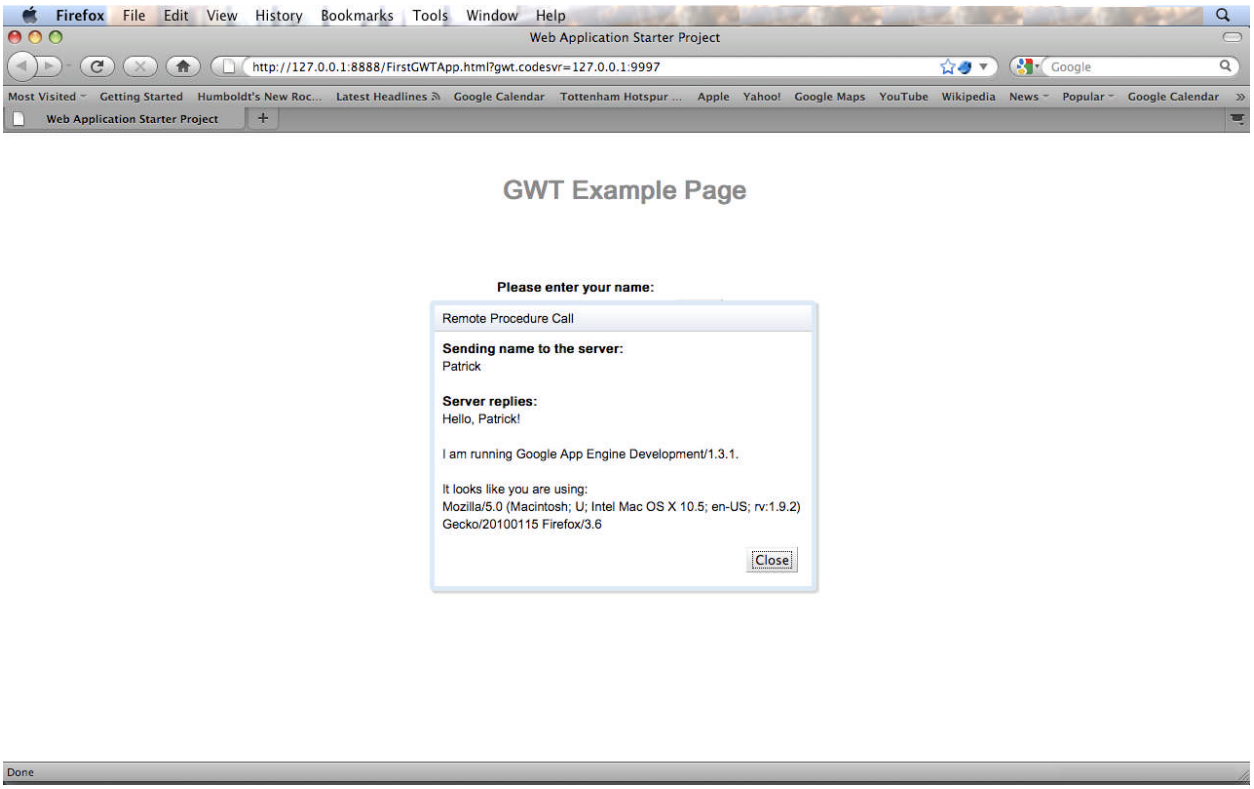

Figure 8 : DialogBox Display of Remote Procedure Call

The source code for FieldVerifier.java and GreetingServiceImpl.java can be found in Appendix E.

6 Summary

The purpose of this paper was to explain the process of creating a GWT application using the Google Plugin for Eclipse. The paper explained in detail the process of implementing Runnable GWT Widgets in a simple GWT module.

The application designed in accordance with this paper could be deployed to the Google App Engine, a free service provided by Google for developers of GWT applications. The application could then be used by any user with a web browser. For more information on deploying GWT applications, visit: http://code.google.com/webtoolkit/doc/latest/tutorial/appengine.html

Works Consulted

"Cascading Style Sheets -." *Wikipedia, the Free Encyclopedia*. Web. 23 Mar. 2010. <http://en.wikipedia.org/wiki/CSS>.

"Google Plugin for Eclipse 3.5 (Galileo) Installation Instructions - Google Plugin for Eclipse -." *Google Code*. Web. 16 Mar. 2010. <http://code.google.com/eclipse/docs/install-eclipse-3.5.html>.

"Google Web Toolkit -." *Wikipedia, the Free Encyclopedia*. Wikipedia. Web. 13 Mar. 2010. <http://en.wikipedia.org/wiki/Google_Web_Toolkit>.

"Google Web Toolkit Overview - Google Web Toolkit -." *Google Code*. Web. 16 Mar. 2010. <http://code.google.com/webtoolkit/overview.html>.

"Using the Google Plugin for Eclipse - Google App Engine -." *Google Code*. Web. 16 Mar. 2010. <http://code.google.com/appengine/docs/java/tools/eclipse.html>.

"Widget Gallery." *Google Code*. Web. 23 Mar. 2010. <http://code.google.com/docreader/#p=google-web-toolkit-doc-1-5&s=google-web-toolkit-doc-1-5&t=DevGuideWidgetGallery>.

Appendix A: Java Source Code

FirstGWTApp.java

```
package com.firstgwtapp.client;

import com.firstgwtapp.shared.FieldVerifier;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.event.dom.client.KeyCodes;
import com.google.gwt.event.dom.client.KeyUpEvent;
import com.google.gwt.event.dom.client.KeyUpHandler;
import com.google.gwt.user.client.rpc.AsyncCallback;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.DialogBox;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.Label;
//import com.google.gwt.user.client.ui.MultiWordSuggestOracle;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TextBox;
import com.google.gwt.user.client.ui.VerticalPanel;
import com.google.gwt.user.client.ui.PopupPanel;
import com.google.gwt.user.client.Window;




/**
 * Entry point classes define <code>onModuleLoad()</code>.
 */
public class FirstGWTApp implements EntryPoint {
	/**
	 * The message displayed to the user when the server cannot be reached
or
	 * returns an error.
	 */
	private static final String SERVER_ERROR = "An error occurred while "
			+ "attempting to contact the server. Please check your
network "
			+ "connection and try again.";

	/**
	 * Create a remote service proxy to talk to the server-side Greeting
service.
	 */
	private final GreetingServiceAsync greetingService = GWT
			.create(GreetingService.class);

	/**
	 * This is the entry point method.
	 */
	public void onModuleLoad() {
```

```
            final Button sendButton = new Button("Send");

            final TextBox nameField = new TextBox();
            final Label errorLabel = new Label();


            nameField.setText("GWT User");
            // We can add style names to widgets
            sendButton.addStyleName("sendButton");

            // Add the nameField and sendButton to the RootPanel
            // Use RootPanel.get() to get the entire body element
            RootPanel.get("nameFieldContainer").add(nameField);
            RootPanel.get("sendButtonContainer").add(sendButton);
            RootPanel.get("runButtonContainer").add(new
RunButton("Runnable!"){
                  public void run(){
                        final PopupPanel popupPanel = new PopupPanel(true);
                    popupPanel.setPopupPositionAndShow(new
PopupPanel.PositionCallback() {
                        public void setPosition(int offsetWidth, int
offsetHeight) {
                          int left = (Window.getClientWidth() - offsetWidth)
/ 3;
                          int top = (Window.getClientHeight() - offsetHeight)
/ 3;
                          popupPanel.setPopupPosition(left, top);
                        }
                      });
                    popupPanel.setWidget(new Label("This is a pop-up from a
RunButton"));
                    popupPanel.show();

                    } // end run method

                  } // ends instantiation of RunButton

            ); // add RunButtonContainer to RootPanel


            RootPanel.get("errorLabelContainer").add(errorLabel);
            RootPanel.get("runRadioButtonContainer").add(new
RunRadioButton("RunRadioButton","Runnable!",true){
                  public void run(){
                        final PopupPanel popupPanel = new PopupPanel(true);
                    popupPanel.setPopupPositionAndShow(new
PopupPanel.PositionCallback() {
                        public void setPosition(int offsetWidth, int
offsetHeight) {
                          int left = (Window.getClientWidth() - offsetWidth)
/ 3;
                          int top = (Window.getClientHeight() - offsetHeight)
/ 3;
                          popupPanel.setPopupPosition(left, top);
                        }
                      });
```

```
                    popupPanel.setWidget(new Label("This is a pop-up from a
RunRadioButton"));
                    popupPanel.show();

                    } // end run method

                  } // ends instantiation of RunRadioButton


                  ); // add RunRadioButtonContainer to RootPanel
            RootPanel.get("runCheckBoxContainer").add(new
RunCheckBox("Runnable!"){
            public void run(){
                  final PopupPanel popupPanel = new PopupPanel(true);
              popupPanel.setPopupPositionAndShow(new
PopupPanel.PositionCallback() {
                  public void setPosition(int offsetWidth, int offsetHeight)
{
                    int left = (Window.getClientWidth() - offsetWidth) / 3;
                    int top = (Window.getClientHeight() - offsetHeight) / 3;
                    popupPanel.setPopupPosition(left, top);
                  }
                });
              popupPanel.setWidget(new Label("This is a pop-up from a
RunCheckBox"));
              popupPanel.show();

              } // end run method

            } // ends instantiation of RunCheckBox


            ); // add RunCheckBoxContainer to RootPanel

            //RootPanel.get("textAreaContainer").add(textArea);
            RootPanel.get("runListBoxContainer").add(new RunListBox(){

                  public void run(){
                        final PopupPanel popupPanel = new PopupPanel(true);
                    popupPanel.setPopupPositionAndShow(new
PopupPanel.PositionCallback() {
                        public void setPosition(int offsetWidth, int
offsetHeight) {
                          int left = (Window.getClientWidth() - offsetWidth)
/ 3;
                          int top = (Window.getClientHeight() - offsetHeight)
/ 3;
                          popupPanel.setPopupPosition(left, top);
                        }
                      });
                    popupPanel.setWidget(new Label("This is a pop-up from a
RunListBox"));
                    popupPanel.show();

                    } // end run method

                  } // ends instantiation of RunListBox
```

```
                  ); // add RunListBoxContainer to RootPanel

            RootPanel.get("runPasswordTextBoxContainer").add(new
RunPasswordTextBox(){

                  public void run(){
                        final PopupPanel popupPanel = new PopupPanel(true);
                    popupPanel.setPopupPositionAndShow(new
PopupPanel.PositionCallback() {
                        public void setPosition(int offsetWidth, int
offsetHeight) {
                          int left = (Window.getClientWidth() - offsetWidth)
/ 2;
                          int top = (Window.getClientHeight() - offsetHeight)
/ 2;
                          popupPanel.setPopupPosition(left, top);
                        }
                      });
                    popupPanel.setWidget(new Label("You're typing in a
RunPasswordTextBox!"));
                    popupPanel.show();

                    } // end run method

                  } // ends instantiation of RunPasswordTextBox


                  ); // add RunPasswordTextBoxContainer to RootPanel

            RootPanel.get("runTextAreaContainer").add(new RunTextArea(){

                  public void run(){
                        final PopupPanel popupPanel = new PopupPanel(true);
                    popupPanel.setPopupPositionAndShow(new
PopupPanel.PositionCallback() {
                        public void setPosition(int offsetWidth, int
offsetHeight) {
                          int left = (Window.getClientWidth() - offsetWidth)
/ 2;
                          int top = (Window.getClientHeight() - offsetHeight)
/ 2;
                          popupPanel.setPopupPosition(left, top);
                        }
                      });
                    popupPanel.setWidget(new Label("You're typing in a
RunTextArea!"));
                    popupPanel.show();

                    } // end run method

                  } // ends instantiation of RunTextArea


                  ); // add RunPasswordTextBoxContainer to RootPanel
```

```
/*
               MultiWordSuggestOracle oracle = new MultiWordSuggestOracle();
               oracle.add("Fairfield");
               oracle.add("Greenwich");
               oracle.add("New Haven");
               oracle.add("Hartford");
               oracle.add("Bridgeport");
               oracle.add("New Hartford");

            RootPanel.get("runSuggestBoxContainer").add(new
RunSuggestBox(oracle){

                  public void run(){
                        final PopupPanel popupPanel = new PopupPanel(true);
                    popupPanel.setPopupPositionAndShow(new
PopupPanel.PositionCallback() {
                        public void setPosition(int offsetWidth, int
offsetHeight) {
                          int left = (Window.getClientWidth() - offsetWidth)
/ 2;
                          int top = (Window.getClientHeight() - offsetHeight)
/ 2;
                          popupPanel.setPopupPosition(left, top);
                        }
                      });
                    popupPanel.setWidget(new Label("This is a
RunSuggestBox"));
                    popupPanel.show();

                    } // end run method

                  } // ends instantiation of RunPasswordTextBox


                  ); // add RunPasswordTextBoxContainer to RootPanel
*/

            // Focus the cursor on the name field when the app loads
            nameField.setFocus(true);
            nameField.selectAll();

            // Create the popup dialog box
            final DialogBox dialogBox = new DialogBox();
            dialogBox.setText("Remote Procedure Call");
            dialogBox.setAnimationEnabled(true);
            final Button closeButton = new Button("Close");
            // We can set the id of a widget by accessing its Element
            closeButton.getElement().setId("closeButton");
            final Label nameFieldTextToServerLabel = new Label();
            final HTML serverResponseLabel = new HTML();
            VerticalPanel dialogVPanel = new VerticalPanel();
            dialogVPanel.addStyleName("dialogVPanel");
            dialogVPanel.add(new HTML("<b>Sending name to the server:</b>"));
            dialogVPanel.add(nameFieldTextToServerLabel);
            dialogVPanel.add(new HTML("<br><b>Server replies:</b>"));
            dialogVPanel.add(serverResponseLabel);
```

```
            dialogVPanel.setHorizontalAlignment(VerticalPanel.ALIGN_RIGHT);
            dialogVPanel.add(closeButton);
            dialogBox.setWidget(dialogVPanel);

            // Add a handler to close the DialogBox
            closeButton.addClickHandler(new ClickHandler() {
                  public void onClick(ClickEvent event) {
                        dialogBox.hide();
                        sendButton.setEnabled(true);
                        sendButton.setFocus(true);
                  }
            });

            // Create a handler for the sendButton and nameField
            class MyHandler implements ClickHandler, KeyUpHandler {
                  /**
                   * Fired when the user clicks on the sendButton.
                   */
                  public void onClick(ClickEvent event) {
                        sendDataToServer();

                  }

                  /**
                   * Fired when the user types in the nameField.
                   */
                  public void onKeyUp(KeyUpEvent event) {
                        if (event.getNativeKeyCode() == KeyCodes.KEY_ENTER) {
                              sendDataToServer();
                        }
                  }

                  /**
                   * Send the name from the nameField to the server and wait
for a response.
                   */
                  private void sendDataToServer() {
                        // First, we validate the input.
                        errorLabel.setText("");
                        String nameFieldTextToServer = nameField.getText();
                        if
(!FieldVerifier.isValidName(nameFieldTextToServer)) {
                              errorLabel.setText("Please enter at least four
characters");
                              return;
                        }
            //
      if(!CheckBoxVerifier.isValidCheckBoxes(checkbox1status,checkbox2status)
){
                  //          error.Label
                        //}

                        // Then, we send the input to the server.
                        sendButton.setEnabled(false);

      nameFieldTextToServerLabel.setText(nameFieldTextToServer);
                        serverResponseLabel.setText("");
```

```
                        greetingService.greetServer(nameFieldTextToServer,
                                    new AsyncCallback<String>() {
                                          public void onFailure(Throwable
caught) {
                                                // Show the RPC error message
to the user
                                                dialogBox
                                                            .setText("Remote
Procedure Call - Failure");
                                                serverResponseLabel

      .addStyleName("serverResponseLabelError");

      serverResponseLabel.setHTML(SERVER_ERROR);
                                                dialogBox.center();
                                                closeButton.setFocus(true);
                                          }

                                          public void onSuccess(String
result) {
                                                dialogBox.setText("Remote
Procedure Call");
                                                serverResponseLabel

      .removeStyleName("serverResponseLabelError");

      serverResponseLabel.setHTML(result);
                                                dialogBox.center();
                                                closeButton.setFocus(true);
                                          }
                                    });
                  }
            }

            // Add a handler to send the name to the server
            MyHandler handler = new MyHandler();
            sendButton.addClickHandler(handler);
            nameField.addKeyUpHandler(handler);
      }


}
```

RunButton.java

```
package com.firstgwtapp.client;

import com.google.gwt.user.client.ui.Button;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;


public abstract class RunButton extends Button implements ClickHandler,
Runnable{
```

```
	public RunButton(String label) {
	this.setText(label);
	addClickHandler(this);
		}

		public void onClick(ClickEvent event) {
			run();

		} // end onClick


		}
```

RunCheckBox.java

```
package com.firstgwtapp.client;

import com.google.gwt.user.client.ui.CheckBox;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;


public abstract class RunCheckBox extends CheckBox implements ClickHandler,
Runnable{

	public RunCheckBox(String label) {
		this.setText(label);
		addClickHandler(this);
			}

	public void onClick(ClickEvent event) {
		run();

	} // end onClick


}
```

RunListBox.java

```
package com.firstgwtapp.client;

import com.google.gwt.event.dom.client.ChangeEvent;
import com.google.gwt.event.dom.client.ChangeHandler;
import com.google.gwt.user.client.ui.ListBox;

public abstract class RunListBox extends ListBox implements ChangeHandler,
Runnable{


	public RunListBox() {
		addItem("Massachusetts");
		addItem("New Hampshire");
		addItem("Connecticut");
```

```
            addItem("Maine");
            addItem("Rhode Island");
            addItem("Vermont");
            addChangeHandler(this);
                  }

                  public void onChange(ChangeEvent event) {
                        run();

                  } // end onClick


}
```

RunPasswordTextBox.java

```
package com.firstgwtapp.client;

import com.google.gwt.event.dom.client.KeyUpEvent;
import com.google.gwt.event.dom.client.KeyUpHandler;
import com.google.gwt.user.client.ui.PasswordTextBox;

public abstract class RunPasswordTextBox extends PasswordTextBox implements
KeyUpHandler, Runnable {


      public RunPasswordTextBox() {
            setText("RunPasswordTextBox");
            addKeyUpHandler(this);
                  }

                  public void onKeyUp(KeyUpEvent event) {
                        run();

                  } // end onKeyUp


}
```

RunRadioButton.java

```
package com.firstgwtapp.client;

import com.google.gwt.user.client.ui.RadioButton;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;


public abstract class RunRadioButton extends RadioButton implements
ClickHandler, Runnable{

      public RunRadioButton(String name,String label, Boolean asHTML){
```

```
            super(name);
            addClickHandler(this);
                  }
      public void onClick(ClickEvent event) {
            run();

      } // end onClick

}
```

RunTextArea.java

```
package com.firstgwtapp.client;

import com.google.gwt.event.dom.client.KeyDownEvent;
import com.google.gwt.event.dom.client.KeyDownHandler;
import com.google.gwt.user.client.ui.TextArea;

public abstract class RunTextArea extends TextArea implements KeyDownHandler,
Runnable {

      public RunTextArea() {
            setText("RunTextArea");
            addKeyDownHandler(this);
                  }

                  public void onKeyDown(KeyDownEvent event) {
                        run();

                  } // end onKeyDown

}
```

RunSuggestBox.java *(Note: not implemented in application, only tested)*

```
package com.firstgwtapp.client;

import com.google.gwt.user.client.ui.MultiWordSuggestOracle;
import com.google.gwt.user.client.ui.SuggestBox;
import com.google.gwt.event.dom.client.MouseOverEvent;
import com.google.gwt.event.dom.client.MouseOverHandler;


public abstract class RunSuggestBox extends SuggestBox implements
MouseOverHandler, Runnable {



      public RunSuggestBox(MultiWordSuggestOracle oracle) {
            //this.setText(label);
```

```
            setVisible(true);
            setText("Enter a search");
            //addClickHandler(this);

      }

                  public void onMouseOver(MouseOverEvent event) {
                        run();

                  } // end onClick



}
```

Appendix B: Module XML

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <module rename-to="firstgwtapp">
- <!--  Inherit the core Web Toolkit stuff.
  -->
  <inherits name="com.google.gwt.user.User" />
- <!--  Inherit the default GWT style sheet.  You can change
  -->
- <!--  the theme of your GWT application by uncommenting
  -->
- <!--  any one of the following lines.
  -->
  <inherits name="com.google.gwt.user.theme.standard.Standard" />
- <!--  <inherits name='com.google.gwt.user.theme.chrome.Chrome'/>
  -->
- <!--  <inherits name='com.google.gwt.user.theme.dark.Dark'/>
  -->
- <!--  Other module inherits
  -->
- <!--  Specify the app entry point class.
  -->
  <entry-point class="com.firstgwtapp.client.FirstGWTApp" />
- <!--  Specify the paths for translatable code
  -->
  <source path="client" />
  <source path="shared" />
  </module>
```

Appendix C: Host Page

```
<!doctype html>
<!-- The DOCTYPE declaration above will set the    -->
<!-- browser's rendering engine into               -->
<!-- "Standards Mode". Replacing this declaration  -->
<!-- with a "Quirks Mode" doctype may lead to some -->
<!-- differences in layout.                        -->

<html>
  <head>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">

    <!--                                                                -->
    <!-- Consider inlining CSS to reduce the number of requested files -->
    <!--                                                                -->
    <link type="text/css" rel="stylesheet" href="FirstGWTApp.css">

    <!--                                           -->
    <!-- Any title is fine                         -->
    <!--                                           -->
    <title>Web Application Starter Project</title>

    <!--                                           -->
    <!-- This script loads your compiled module.   -->
    <!-- If you add any GWT meta tags, they must   -->
    <!-- be added before this line.                -->
    <!--                                           -->
    <script type="text/javascript" language="javascript"
src="firstgwtapp/firstgwtapp.nocache.js"></script>
  </head>

  <!--                                         -->
  <!-- The body can have arbitrary html, or    -->
  <!-- you can leave the body empty if you want  -->
  <!-- to create a completely dynamic UI.      -->
  <!--                                         -->
  <body>

    <!-- OPTIONAL: include this if you want history support -->
    <iframe src="javascript:''" id="__gwt_historyFrame" tabIndex='-1'
style="position:absolute;width:0;height:0;border:0"></iframe>

    <!-- RECOMMENDED if your web app will not function without JavaScript
enabled -->
    <noscript>
      <div style="width: 22em; position: absolute; left: 50%; margin-left: -
11em; color: red; background-color: white; border: 1px solid red; padding:
4px; font-family: sans-serif">
        Your web browser must have JavaScript enabled
        in order for this application to display correctly.
      </div>
    </noscript>

    <h1>GWT Example Page</h1>
```

```
    <table align="center">
      <tr>
        <td colspan="2" style="font-weight:bold;">Please enter your
name:</td>
      </tr>
      <tr>
        <td id="nameFieldContainer"></td>
        <td id="sendButtonContainer"></td>
        </tr>
      <tr>
        <td id="runButtonContainer"></td>
      </tr>
        <tr>
        <td id="runRadioButtonContainer"></td>
        <td id="runCheckBoxContainer"></td>
        </tr>
        <tr>
        <td id="runListBoxContainer"></td>
        </tr>
        <tr>
        <td id="runPasswordTextBoxContainer"></td>
        </tr>
        <tr>
        <td id="runTextAreaContainer"></td>

      </tr>
      <tr>
        <td colspan="2" style="color:red;" id="errorLabelContainer"></td>
      </tr>
    </table>
  </body>
</html>
```

Appendix D: Application Style Sheet

```
/** Add css rules here for your application. */


/** Example rules used by the template application (remove for your app) */
h1 {
  font-size: 2em;
  font-weight: bold;
  color: #777777;
  margin: 40px 0px 70px;
  text-align: center;
}

.sendButton {
  display: block;
  font-size: 16pt;
}

/** Most GWT widgets already have a style name defined */
.gwt-DialogBox {
  width: 400px;
}

.dialogVPanel {
  margin: 5px;
}

.serverResponseLabelError {
  color: red;
}

/** Set ids using widget.getElement().setId("idOfElement") */
#closeButton {
  margin: 15px 6px 6px;
}

.gwt-SuggestBox
{
    border                      :    1px solid #666;
    text-align                  :    left;
    width                       :    400px;
}
.gwt-SuggestBoxPopup
{
    text-align                   :    left;
    cursor                       :    pointer;
    cursor                       :    hand;
    border                       :    1px solid #666;
    border-top                   :    0;
    background-color             :    #fff;

}
.gwt-SuggestBoxPopup .item
{
```

```
    border                      :    1px dotted #aaa;
    width                       :    398px;
}
.gwt-SuggestBoxPopup .item-selected
{
    background-color            :    #ffc;
}
```

Appendix E: FieldVerifier.java and GreetingServiceImpl.java

FieldVerifier.java

```
package com.firstgwtapp.shared;

/**
 * <p>
 * FieldVerifier validates that the name the user enters is valid.
 * </p>
 * <p>
 * This class is in the <code>shared</code> packing because we use it in both
 * the client code and on the server. On the client, we verify that the name
is
 * valid before sending an RPC request so the user doesn't have to wait for a
 * network round trip to get feedback. On the server, we verify that the name
is
 * correct to ensure that the input is correct regardless of where the RPC
 * originates.
 * </p>
 * <p>
 * When creating a class that is used on both the client and the server, be
sure
 * that all code is translatable and does not use native JavaScript. Code
that
 * is note translatable (such as code that interacts with a database or the
file
 * system) cannot be compiled into client side JavaScript. Code that uses
native
 * JavaScript (such as Widgets) cannot be run on the server.
 * </p>
 */
public class FieldVerifier {

      /**
       * Verifies that the specified name is valid for our service.
       *
       * In this example, we only require that the name is at least four
       * characters. In your application, you can use more complex checks to
ensure
       * that usernames, passwords, email addresses, URLs, and other fields
have the
       * proper syntax.
       *
       * @param name the name to validate
       * @return true if valid, false if invalid
       */
      public static boolean isValidName(String name) {
            if (name == null) {
                  return false;
            }
            return name.length() > 3;
```

```
	}
}
```

GreetingServiceImpl.java

```
package com.firstgwtapp.server;

import com.firstgwtapp.client.GreetingService;
import com.firstgwtapp.shared.FieldVerifier;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

/**
 * The server side implementation of the RPC service.
 */
@SuppressWarnings("serial")
public class GreetingServiceImpl extends RemoteServiceServlet implements
            GreetingService {

      public String greetServer(String input) throws IllegalArgumentException
{
            // Verify that the input is valid.
            if (!FieldVerifier.isValidName(input)) {
                  // If the input is not valid, throw an
IllegalArgumentException back to
                  // the client.
                  throw new IllegalArgumentException(
                              "Name must be at least 4 characters long");
            }

            String serverInfo = getServletContext().getServerInfo();
            String userAgent = getThreadLocalRequest().getHeader("User-
Agent");
            return "Hello, " + input + "!<br><br>I am running " + serverInfo
                        + ".<br><br>It looks like you are using:<br>" +
userAgent;
      }
}
```