Google Web Toolkit (GWT)

Mark Volkmann
mark@ociweb.com

What Is GWT?

- An open-source, Java-based framework for creating Ajax web applications
- Created and used by Google
- Makes writing web applications similar to Swing applications
  - dramatically reduces the need to understand HTML and JavaScript
  - maintaining Java code is easier than maintaining a mixture of HTML, JavaScript and Java code
- Client-side code
  - compiled to HTML and JavaScript
  - uses CSS for formatting
  - restricted to a subset of Java 1.4 for now
- Server-side code can be implemented in any language
  - including Java 5 and 6
  - commonly Java-based GWT RPC servlets are used (more on this later)

Why Use GWT?

- Creates browser-based GUIs using Swing-like components
- No need to write HTML which means no scriptlets
- Uses CSS for formatting and some layout
- No need to write JavaScript, but can if desired
- No messy navigation control with redirects and forwards
- Makes asynchronous server calls
  which results in a better user experience
- Ajax features can be implemented
  without thinking about DOM manipulation
- Direct DOM manipulation, if needed, is easy to do
- Can easily pass Java objects between client and server
  - no need to populate Java objects from HTTP data
  - no need to populate HTML forms from Java objects
- Can throw Java exceptions from the server back to the client
- Can use Java IDEs like Eclipse, IDEA and NetBeans
- Created, maintained, enhanced and used by Google
  - recognized for their software expertise
  - one of the few serious Microsoft competitors

Tool Support

- IDE plugins
  - Intellij IDEA
    - GWT Studio
  - Eclipse
    - Cypal Studio for GWT (was Googlipse)
    - Instantiations GWT Designer
  - NetBeans
    - GWT4NB
- Other tools
  - Wirelexsoft VistaFei - a GWT IDE

Installing

- Steps
  - verify that a Java SDK (1.4 or higher) is installed
    - run "`java -version`" from a shell/command prompt
  - browse http://code.google.com/webtoolkit/
  - click the "Download Google Web Toolkit (GWT)" link
  - download a platform-specific archive/zip file for a selected version
  - unpackage the archive/zip file to a selected directory
  - set the environment variable `GWT_HOME` to the path to the unpackaged GWT directory
- Can also build latest code from source
  - see http://code.google.com/webtoolkit/makinggwtbetter.html#compiling

Creating a New Project

- Steps differ
  - depending on whether an IDE plugin is being used
  - we'll assume no IDE
- Non-IDE steps
  - create a directory for the application
  - from a shell/command prompt, navigate to that directory
  - use the **`applicationCreator`** script to create the project directory structure and populate it with initial project files
    - under Windows, run
      `%GWT_HOME%\applicationcreator {`*`package`*`}.client.{`*`module`*`}`
    - under UNIX/Linux/Mac OS X, run
      `$GWT_HOME/applicationcreator {`*`package`*`}.client.{`*`module`*`}`

Typical Project Directory Structure

- **Ant** build.properties and build.xml
- **build**
  - **classes** - holds .class files generated by Java compiler
  - **www** - holds HTML, CSS, images and JavaScript (including that generated by GWTCompiler)
- **lib** - holds JARs used by this project
- **src**
  - *package* directories
    - ***module*.gwt.xml**
    - **client**
      - *module*.java entry point class
      - *service*.java
      - *service*Async.java
      - other .java files to be compiled to JavaScript
    - **public**
      - *module*.html
      - *module*.css
      - images
    - **server** - not automatically created
      - *service*Impl.java
      - other server-side only .java files
- **tomcat** - used by hosted mode embedded server

```
{package}.{module}.cache.html
{hash-code}.nocache.html
{hash-code}.nocache.xml
```

for a GWT RPC service

Implementing a New Project

- The initial files to be edited for a new project are
  - the main HTML file
    - `src/{package-dirs}/public/{module}.html`
  - the entry point Java source file
    - `src/{package-dirs}/client/{module}.java`

Minimal HTML

```
<html>
  <head>
    <title>{title}</title>
    <meta name="gwt:module" content="{package}.{module}">
  </head>
  <body>
    <script language="javascript" src="gwt.js"></script>

    <!-- OPTIONAL: include this if history support is desired -->
    <iframe id="__gwt_historyFrame" style="width:0;height:0;border:0">
    </iframe>

    <!-- can add static HTML here -->
  </body>
</html>
```

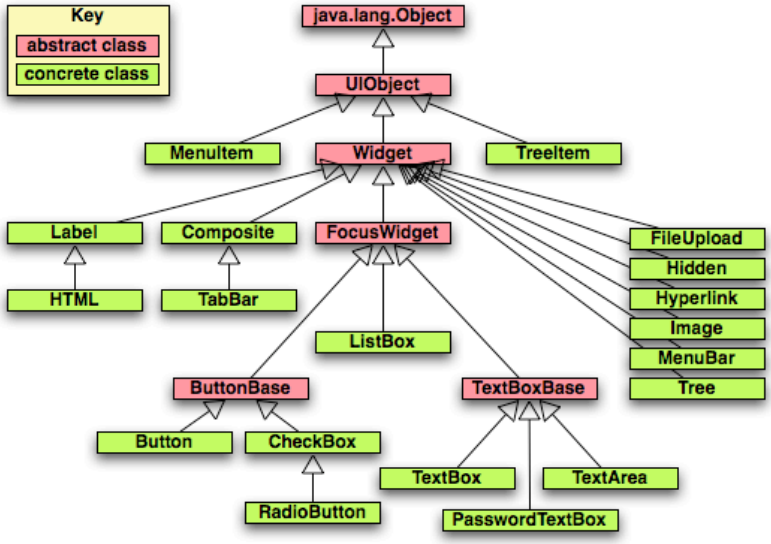
to find module XML

"bootstrap script" loads browser/local-specific JavaScript

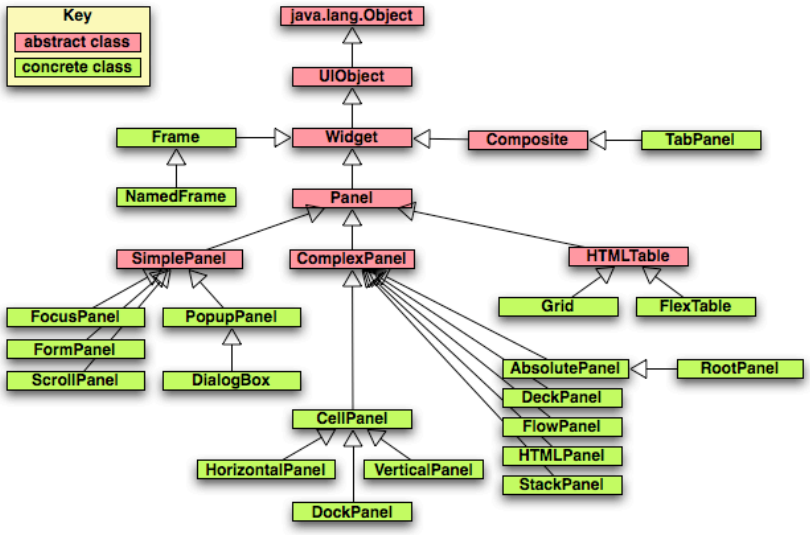
Minimal Entry Point Java Class

```
package com.ociweb.demo.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.*;

public class {module} implements EntryPoint {
  public void onModuleLoad() {
    final TextBox textBox = new TextBox();
    Button button = new Button("Click me");
    button.addClickListener(new ClickListener() {
      public void onClick(Widget sender) {
        textBox.setText("clicked");
      }
    });
    Panel panel = new FlowPanel();
    panel.add(textBox);
    panel.add(button);
    RootPanel.get().add(panel);
  }
}
```

Widgets



Containers



Modules

- GWT applications are referred to as “modules”
- A module has an XML descriptor, {*module*}.gwt.xml
  - can specify
    - inherited modules
      - to share custom widgets and other code
    - entry point Java class name
    - source paths
      - for locating Java source files to be compiled to JavaScript
    - public paths
      - for locating CSS, JavaScript and image files
    - and more
  - stored in the source tree package directory just above entry point Java source file

Example Module XML

```
<module>
  <inherits name="com.google.gwt.user.User"/>

  <source path="client"/> <!-- the default -->

  <entry-point class="com.ociweb.gwt.client.Hello"/>

  <stylesheet src="Hello.css"/>

  <servlet path="/com.ociweb.gwt.Hello/SomeService"
    class="com.ociweb.gwt.server.SomeServiceImpl"/>
</module>
```

module package

module name

For details on the content of module XML files, see
http://code.google.com/webtoolkit/documentation/
com.google.gwt.doc.DeveloperGuide.Fundamentals.Modules.ModuleXml.html

CSS

- GWT encourages formatting with CSS
- All widgets support these methods
  - `addStyleName` - adds to existing list of styles for the widget
  - `setStyleName` - replaces existing list of styles for the widget
- To specify formatting for each style name
  - add a CSS class to a stylesheet file (.css) in the `src/{package-dirs}/public` directory
  - add the following line to the module XML
    `<stylesheet src="{name}.css"/>`
- Most widgets have a default CSS class name
  - for example, Button widgets use the `gwt-Button` CSS class
- For more info, see
  http://www.ociweb.com/mark/programming/GWT.html#Formatting

Event Handling

- Most widgets support listening for user events
  - events generated by each widget differ
- Listener interfaces
  - define methods that are invoked on objects that implement them (your code) when certain events occur
  - for example, ClickListener
  - often implemented by an anonymous inner class
    - like earlier example
- Adapter classes
  - make it easier to implement listener interfaces by providing method implementations that do nothing, saving you the trouble of writing them
  - only for interfaces with more than one method
- For more info. see
  - http://www.ociweb.com/mark/programming/GWT.html#Widgets
  - http://www.ociweb.com/mark/programming/GWT.html#ListenersAdapters

Event Handling (Cont'd)

- Listener interfaces and adapter classes are summarized below

Name	Adapter	Methods
ChangeListener	none	`void onChange(Widget sender)`
ClickListener	none	`void onClick(Widget sender)`
FocusListener	FocusListenerAdapter	`void onFocus(Widget sender)` `void onLostFocus(Widget sender)`

KeyboardListener	KeyboardListenerAdapter	`void onKeyDown(Widget sender, char keyCode, int modifiers)`
		`void onKeyPress(Widget sender, char keyCode, int modifiers)`
		`void onKeyUp(Widget sender, char keyCode, int modifiers)`

LoadListener	none	`void onError(Widget sender)` `void onLoad(Widget sender)`

MouseListener	MouseListenerAdapter	`void onMouseDown(Widget sender, int x, int y)`
		`void onMouseEnter(Widget sender)`
		`void onMouseLeave(Widget sender)`
		`void onMouseMove(Widget sender, int x, int y)`
		`void onMouseUp(Widget sender, int x, int y)`

MouseWheelListener	none	`void onMouseWheel(Widget sender, int x, int y, MouseWheelVelocity velocity)`
PopupListener	none	`void onPopupClosed(PopupPanel sender, boolean autoClosed)`
ScrollListener	none	`void onScroll(Widget sender, int scrollLeft, int scrollTop)`
TableListener	none	`void onCellClicked(SourcedTableEvents sender, int row, int cell)`
TabListener	none	`void onBeforeTabSelected(SourcesTabEvents sender, int tabIndex)` `void onTabSelected(SourcesTabEvents sender, int tabIndex)`
TreeListener	none	`void onTreeItemStateChanged(TreeItem item)` `void onTreeItemStateChanged(TreeItem item)`

Running in Hosted Mode

- In "hosted mode", a special browser is used to display and exercise the user interface
  - uses Java bytecode instead of generated JavaScript
- Once the hosted mode browser is running
  - changes to client-side Java code (made in any editor) can be tested by simply clicking the refresh button in the browser
  - no separate compiling or deploying is required!
  - allows quick testing of CSS and client-side Java code changes
  - can use an IDE debugger to step through code
- Server-side code changes still require recompile/redeploy
- Steps
  - from a shell/command prompt, navigate to application directory
  - run `{module}-shell` script
    - this compiles client-side code, starts hosted mode browser, and runs application inside it
  - can also launch from Ant or an IDE

`-noserver` option allows use a server other than embedded Tomcat

Running in Deployed Mode

- To deploy a GWT application to an app. server so it can be run outside hosted mode
  - compile client-side Java code into HTML and JavaScript using the GWTCompiler

    By default, JavaScript is generated in "obfuscate" mode which results in smaller files. Another supported mode is "pretty" which makes the files readable, but larger. The last mode is "detailed" which is like "pretty", but adds more verbose names to assist with tracing JavaScript functions back to methods in specific Java classes. To specify the mode that the GWT compiler should use, run the compile script with the `-style=OBF|PRETTY|DETAILED` option
  - compile server-side Java code using a standard Java compiler
  - GWTCompiler generates a separate JavaScript file for each supported browser and locale combination
    - the correct JavaScript file is automatically downloaded to the requesting browser by the **`gwt.js`** "bootstrap script"
    - this reduces the amount of JavaScript code that must be downloaded since only code applicable to that browser/locale is downloaded

changing in GWT 1.4

continued on next page

Running in Deployed Mode (Cont'd)

  - bundle the following in a WAR file
    - module XML and HTML
    - generated HTML/JavaScript
    - CSS files
    - image files
    - server-side .class files
    - GWT-supplied `gwt-servlet.jar`
    - other JARs required by server-side code
  - deploy the WAR file to a JEE server such as Tomcat or JBoss
  - run the application from a web browser by visiting `http://{server}:{port}/{package}/{module}.html`
- These steps are typically automated with Ant
  - see http://www.ociweb.com/mark/programming/GWT.html#Deploying for an example Ant build file

for debugging in deployed mode, Firefox using Firebug is recommended

Server Calls

- Several forms are supported
  - HTML form submission
  - XMLHttpRequest (classic Ajax)
  - JavaScript Object Notation (JSON)
  - GWT RPC
- Perhaps the most common is GWT RPC
  - uses Java servlets that extend `RemoteServiceServlet`
  - automatically serializes objects from classes that
    - implement `IsSerializable` (a marker interface)
    - have a no-arg constructor
  - many GWT versions of standard Java classes do this
    - such as `java.util.List` and `java.util.Map`
  - see steps to use at
    http://www.ociweb.com/mark/programming/GWT.html#ServerCalls

JavaScript Native Interface (JSNI)

- Java methods can invoke JavaScript code through JSNI
- Uses special native methods that
  embed JavaScript code in Java source files
- Example

```
private native boolean matches(
  String regExp, String value) /*-{
  var pattern = new RegExp(regExp);
  return value.search(pattern) != -1;
}-*/;
```

JavaScript code

Other Supported Features

- Custom widgets
  - implemented by Java classes that typically extend the GWT `Composite` class
- Reusable modules
  - can be "inherited" by multiple GWT applications
- Internationalization
  - using Java property files
- JUnit testing
- Client-side XML processing

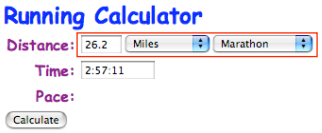
More Information

- Web pages
  - project homepage - http://code.google.com/webtoolkit/
  - developer guide - http://code.google.com/webtoolkit/documentation/
  - class doc. - http://code.google.com/webtoolkit/documentation/gwt.html
  - OCI notes - http://www.ociweb.com/mark/programming/GWT.html
- Mailing list
  - Google Group - http://groups.google.com/group/Google-Web-Toolkit
- Blogs
  - Google Developer Blog - http://googlewebtoolkit.blogspot.com/
  - Robert Hanson (author of "GWT In Action") - http://roberthanson.blogspot.com/
- Podcasts
  - Java Posse "GWT Round Table" on 10/15/06
    - http://javaposse.com/index.php?post_id=140955
- Books
  - none yet, but many on the way; see next page and http://www.ociweb.com/mark/programming/GWT.html#Resources

Books On The Way

Running Calculator Example

- Let's build a web app. for calculating running paces
  - users enter a distance and a time
  - the application calculates the pace per mile
- Here's the GUI we want



- Details
  - part outlined in red is a custom widget called DistanceWidget
    - contains a TextBox for entering a distance
    - contains a ListBox for selecting units of "Miles" or "Kilometers"
      - changing selection converts the distance
    - contains a ListBox for selecting common distances such as "Marathon"
      - selecting one sets the distance TextBox and the units ListBox
  - time TextBox accepts formats of hh:mm:ss, mm:ss or ss

Steps To Write "Entry Point" Class

- We'll assume some things are already in place
  - Calculator.java - does math to convert distances and calculate paces
  - DistanceWidget.java - custom widget for entering a distance
- RunningCalc.java - implements EntryPoint
  - create the following widgets
    - Label to display application title
    - DistanceWidget
    - TextBox to enter time (set visible length to 10)
    - Label to display calculated pace
    - FlexTable to layout labels and data entry/display widgets
    - "Calculate" button
  - add widgets to the FlexTable
  - create VerticalPanel and use it to layout application title, FlexTable and "Calculate" button
  - add VerticalPanel to RootPanel
  - add a ClickListener to "Calculate" button that calculates pace and sets pace Label

```
double miles = distanceWidget.getMiles();
String time = timeTextBox.getText();
String pace = Calculator.getMilePaceTime(miles, time);
paceLabel.setText(pace);
```

Additional Tasks:
- add validation to the distance and time TextBoxes
- add CSS to make the GUI look nicer
- add a KeyboardListener to the time TextBox so pressing Enter there is the same as pressing the "Calculate" button

The Finished Code

- All of the code in the finished application follows

Calculator.java

```
package com.ociweb.running.client;

public class Calculator {

  private static final double KILOMETERS_PER_MILE = 1.609344;
  private static final double MILES_PER_KILOMETER = 1 / KILOMETERS_PER_MILE;

  public static double convertKilometersToMiles(double kilometers) {
    return kilometers * MILES_PER_KILOMETER;
  }

  public static double convertMilesToKilometers(double miles) {
    return miles * KILOMETERS_PER_MILE;
  }
```


Calculator.java (Cont'd)

```
  /**
   * Converts a time string to the equivalent number of seconds.
   * Times must be in the format hh:mm:ss, mm:ss or ss.
   */
  public static int convertTimeToSeconds(String time) {
    // Validate the time.
    String regex = "/(\\d{1,2})(:(\\d{1,2}))?(:(\\d{1,2}))?/";
    if (matches(regex, time)) {
      throw new RuntimeException("Invalid time: " + time);
    }

    String[] pieces = time.split(":");
    int count = pieces.length;
    int p0 = Integer.parseInt(pieces[0]);
    int p1 = count > 1 ? Integer.parseInt(pieces[1]) : 0;
    int p2 = count > 2 ? Integer.parseInt(pieces[2]) : 0;
    int hours = count == 3 ? p0 : 0;
    int minutes = count == 3 ? p1 : count == 2 ? p0 : 0;
    int seconds = count == 3 ? p2 : count == 2 ? p1 : p0;
    return (hours * 60 + minutes) * 60 + seconds;
  }
```

Calculator.java (Cont'd)

```
/**
 * Gets the minutes per mile pace time string
 * for a given distance (in miles) and total time.
 * Times must be in the format hh:mm:ss, mm:ss or ss.
 */
public static String getMilePaceTime(double miles, String time) {
  int seconds = convertTimeToSeconds(time);
  long secondsPerMile = Math.round(seconds / miles);
  double minutesPerMile = secondsPerMile / 60.0;
  int wholeMinutesPerMile = (int) Math.floor(minutesPerMile);
  long wholeSecondsPerMile =
    Math.round((minutesPerMile - wholeMinutesPerMile) * 60.0);
  String pace = wholeMinutesPerMile + ":";
  if (wholeSecondsPerMile < 10) pace = '0' + pace;
  pace += wholeSecondsPerMile + " per mile";
  return pace;
}
```


Calculator.java (Cont'd)

```
/**
 * Gets the minutes per kilometer pace time string
 * for a given distance (in kilometers)
 * and total time.
 * Times must be in the format hh:mm:ss, mm:ss or ss.
 */
public static String getKilometerPaceTime(
  double kilometers, String time) {
  return getMilePaceTime(kilometers, time);
}

/**
 * Regular expression matching using JavaScript (JSNI).
 * @param regExp the regular expression
 * @param value the value to be compared
 * @return true if the value matches; false otherwise
 */
public native static boolean matches(String regExp, String value) /*-{
  var pattern = new RegExp(regExp);
  return value.search(pattern) != -1;
}-*/;
}
```

DistanceWidget.java

```
package com.ociweb.running.client;

import com.google.gwt.user.client.ui.*;
import java.util.Map;
import java.util.HashMap;

public class DistanceWidget extends Composite {
  private static Map distanceMap = new HashMap();

  private ListBox distanceListBox = new ListBox();
  private ListBox unitListBox = new ListBox();
  private TextBox distanceTextBox = new TextBox();
  private int otherIndex;

  public DistanceWidget() {
    setupDistanceTextBox();
    setupUnitListBox();
    setupDistanceListBox();

    Panel panel = new HorizontalPanel();
    panel.add(distanceTextBox);
    panel.add(unitListBox);
    panel.add(distanceListBox);
    initWidget(panel);
  }
```


DistanceWidget.java (Cont'd)

```
  private void addDistance(String name, double distance) {
    distanceMap.put(name, new Double(distance));
    if ("Other".equals(name)) otherIndex = distanceListBox.getItemCount();
    distanceListBox.addItem(name);
  }

  private double getDistance(String name) {
    Double distance = (Double) distanceMap.get(name);
    return distance.doubleValue();
  }

  public double getMiles() {
    double distance = Double.parseDouble(distanceTextBox.getText());
    boolean isKilometers = unitListBox.getSelectedIndex() == 1;
    if (isKilometers) {
      distance = Calculator.convertKilometersToMiles(distance);
    }
    return distance;
  }
```

DistanceWidget.java (Cont'd)

```
private void setupDistanceListBox() {
  addDistance("Marathon", 26.2);
  addDistance("1/2 Marathon", 13.1);
  addDistance("10K", 10);
  addDistance("5K", 5);
  addDistance("Mile", 1);
  addDistance("Other", 0);

  distanceListBox.setSelectedIndex(0);
  distanceTextBox.setText(String.valueOf(getDistance("Marathon")));

  distanceListBox.addChangeListener(new ChangeListener() {
    public void onChange(Widget sender) {
      int index = distanceListBox.getSelectedIndex();
      String name = distanceListBox.getItemText(index);
      double distance = getDistance(name);
      String distanceText = distance == 0 ? "" : String.valueOf(distance);
      distanceTextBox.setText(distanceText);
      unitListBox.setSelectedIndex(name.endsWith("K") ? 1 : 0);
    }
  });
}
```


DistanceWidget.java (Cont'd)

```
private void setupDistanceTextBox() {
  distanceTextBox.setVisibleLength(5);
  distanceTextBox.addFocusListener(new ChangeListener () {
    public void onChange(Widget sender) {
      distanceListBox.setSelectedIndex(otherIndex);
    }
  });
}
```

DistanceWidget.java (Cont'd)

```
  private void setupUnitListBox() {
    unitListBox.addItem("Miles");
    unitListBox.addItem("Kilometers");
    unitListBox.setSelectedIndex(0);

    unitListBox.addChangeListener(new ChangeListener() {
      public void onChange(Widget sender) {
        // Convert value in distanceTextBox to new unit.
        int index = unitListBox.getSelectedIndex();
        String distanceText = distanceTextBox.getText();
        double distance = Double.parseDouble(distanceText);
        distance = index == 0 ?
          Calculator.convertKilometersToMiles(distance) :
          Calculator.convertMilesToKilometers(distance);
        distanceTextBox.setText(String.valueOf(distance));
      }
    });
  }
}
```


RunningCalc.java

```
package com.ociweb.running.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.*;

public class RunningCalc implements EntryPoint {

  private Button calculateButton = new Button("Calculate");
  private DistanceWidget distanceWidget = new DistanceWidget();
  private Label paceLabel = new Label();
  private FlexTable table = new FlexTable();
  private TextBox timeTextBox = new TextBox();
  private int row = 0;
```

RunningCalc.java (Cont'd)

```
  public void onModuleLoad() {
    timeTextBox.setText("2:57:11");
    timeTextBox.setVisibleLength(10);
    timeTextBox.addKeyboardListener(new KeyboardListenerAdapter() {
      public void onKeyUp(Widget sender, char keyCode, int modifiers) {
        if (keyCode == KEY_ENTER) calculatePace();
      }
    });

    paceLabel.addStyleName("paceLabel");

    calculateButton.addClickListener(new ClickListener() {
      public void onClick(Widget sender) {
        calculatePace();
      }
    });

    addRow("Distance:", distanceWidget);
    addRow("Time:", timeTextBox);
    addRow("Pace:", paceLabel);
```


RunningCalc.java (Cont'd)

```
    VerticalPanel panel = new VerticalPanel();
    Label titleLabel = new Label("Running Calculator");
    titleLabel.addStyleName("title");
    panel.add(titleLabel);
    panel.add(table);
    panel.add(calculateButton);

    RootPanel.get().add(panel);
  }

  private void addRow(String labelText, Widget widget) {
    Label label = new Label(labelText);
    label.addStyleName("tableLabel");
    table.setWidget(row, 0, label);
    table.setWidget(row, 1, widget);
    row++;
  }

  private void calculatePace() {
    double miles = distanceWidget.getMiles();
    String time = timeTextBox.getText();
    String pace = Calculator.getMilePaceTime(miles, time);
    paceLabel.setText(pace);
  }
}
```

RunningCalc.gwt.xml

```
<module>

  <inherits name='com.google.gwt.user.User'/>

  <entry-point class='com.ociweb.running.client.RunningCalc'/>

</module>
```


RunningCalc.html

```
<html>
  <head>
    <title>RunningCalc Application</title>
    <meta name='gwt:module' content='com.ociweb.running.RunningCalc'>
    <link rel=stylesheet href="RunningCalc.css">
  </head>
  <body>
    <script language="javascript" src="gwt.js"></script>
    <iframe id="__gwt_historyFrame"
      style="width:0;height:0;border:0"></iframe>
  </body>
</html>
```

RunningCalc.css

```
body {
  font-family: Comic Sans MS, sans-serif;
  font-size: small;
  margin: 8px;
}

.paceLabel {
  color: green;
  font-size: 10pt;
  font-weight: bold;
}

.tableLabel {
  color: purple;
  font-size: 12pt;
  font-weight: bold;
  text-align: right;
}

.title {
  color: blue;
  font-size: 18pt;
  font-weight: bold;
}
```


Wrap-Up

- GWT provides a way to leverage Java expertise and tools to build dynamic web applications without requiring knowledge of HTML, JavaScript and DOM manipulation
- CAIT may have a course on GWT soon
- Feel free to email questions about GWT to me at mark@ociweb.com
- Thanks for attending!