The Google Web Toolkit (GWT): Declarative Layout with UiBinder – Advanced Topics

(GWT 2.5 Version)

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/gwt.html

© 2013 Marty Hall & Yaakov Chaikin

Topics in This Section

- **Quick review of UiBinder steps**
  - HTML-based and Widget-based
- **Using widget attributes in UiBinder XML**
- **Three ways to deal with lack of widget zero-arg constructor in UiBinder XML**
  - Using @UiConstructor
  - Using @UiFactory
  - Using @UiField(provided=true)
- **Layout panels with special UiBinder XML syntax**
- **Working with CSS in UiBinder**
- **Working with images in UiBinder**

Quick Review of UiBinder Steps

HTML-Based (Eclipse Wizard Builds #1 & #2)

1. **Create XML file**
   - Top-level element is an HTML tag like
   - Put raw HTML inside
   - Use ui:field to mark places for later insertion
2. **Create Java class that represents the XML**
   - Extend UiObject
   - Use UiBinder interface and GWT.create to bind the Java class to the XML representation
   - Mark fields with @UiField (one for each ui:field in XML)
   - Call setElement
   - Insert content into those fields
3. **Use new Java class in EntryPoint class**
   - Use DOM API with getElement and appendChild

Widget-Based (Eclipse Wizard Builds #1 & #2)

1. **Create XML file**
   - Top-level element represents a Widget: <g:someGwtWidget>
   - Put mixture of raw HTML and <g:otherGwtWidgets> inside
   - Use ui:field to mark places for later insertion
2. **Create Java class that represents the XML**
   - Extend Composite
   - Use UiBinder interface and GWT.create to bind the Java class to the XML representation
   - Mark fields with @UiField (one for each ui:field in XML)
   - Call initWidget
   - Set properties of those fields
3. **Use new Java class in EntryPoint class**
   - Use normal Widget methods
     - E.g., RootPanel.get(…).add(yourWidget)

Using Widget Attributes in UiBinder

Summary

- **Idea**
  - Most widgets have setters that alter their look or behavior
- **Previously: called in Java**
  - After fields were bound, we called widget setters in Java code
    - Reduces readability of UI XML template
- **New alternative: declared in XML**
  - Use XML attributes that correspond to setter methods
    - Follows JavaBean notation
- **Example**
  - <g:ListBox ui:field="stateList" visibleItemCount="6"/>

Example: Improved States/Cities ListBox ImprovedUi.ui.xml

```
<!DOCTYPE ui:UiBinder SYSTEM
"http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
xmlns:g="urn:import:com.google.gwt.user.client.ui">
<g:HTMLPanel>
<table border="1">
 <tr><th>User Control</th><th>Result</th></tr>
 <tr><td>State:
 <g:ListBox ui:field="stateList" visibleItemCount="6"/>
 </td><td>City:
 <g:ListBox ui:field="cityList"
            visibleItemCount="1" enabled="false"/>
 </td></tr>
</table><br/>
</g:HTMLPanel>
</ui:UiBinder>
```


Example: Improved States/Cities ListBox ImprovedUi.java

```
...
public class ImprovedUi extends Composite {
  private static ImprovedUiUiBinder uiBinder =
                    GWT.create(ImprovedUiUiBinder.class);
  interface ImprovedUiUiBinder
                  extends UiBinder<Widget, ImprovedUi> {
  }

  @UiField ListBox stateList;
  @UiField ListBox cityList;

  public ImprovedUi() {
    initWidget(uiBinder.createAndBindUi(this));
    populateStateList();
    cityList.addItem("Select City");
  }
...
```

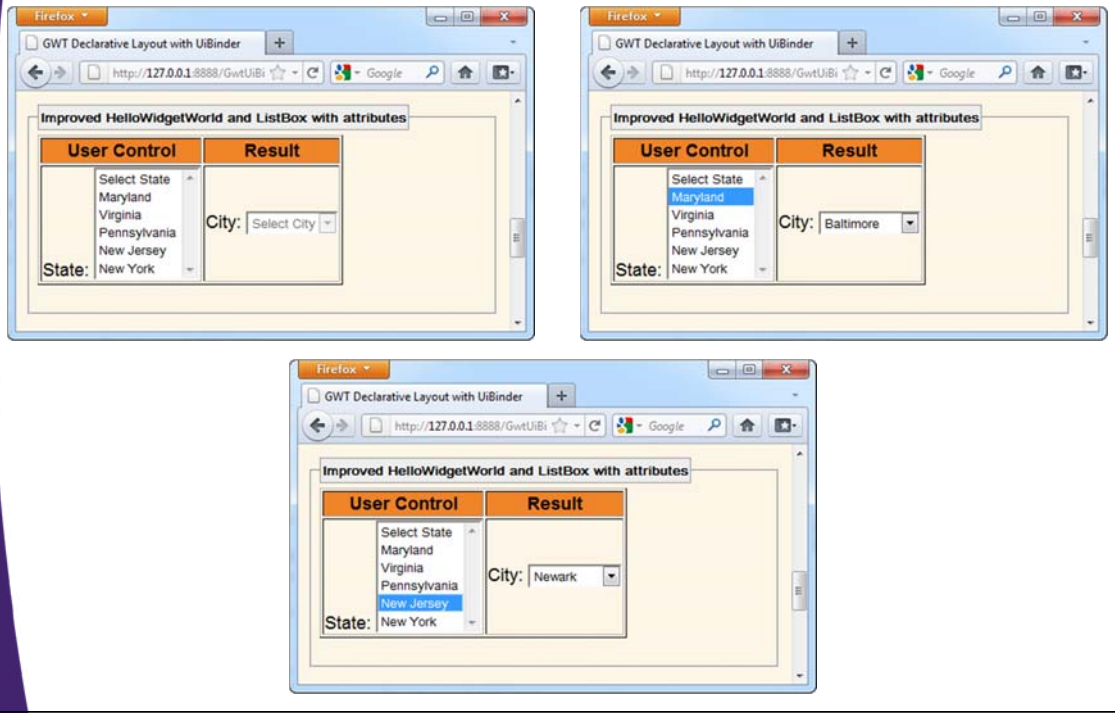
Since all the visual setup is done in UI XML template, no longer a need for setupListBox method. All we need is to do is populate both ListBoxes.

Example: Improved States/Cities ListBox GwtUiBinder1.html & EntryPoint Class

```
...
<fieldset>
<legend>
Improved HelloWidgetWorld and ListBox with attributes
</legend>
<div id="improved-ui"></div>
</fieldset>
```

```
...
public class GwtUiBinder1 implements EntryPoint {

  public void onModuleLoad() {
    ...

    ImprovedUi improvedUi = new ImprovedUi();
    RootPanel.get("improved-ui").add(improvedUi);
    ...
  }
}
```


Example: Improved States/Cities ListBox: Result



Giving Widget Constructor Arguments

Motivation

- **Widgets declared in UI XML template get created by a call to GWT.create()**
  - In most cases, this means the widget must have a default, (zero-arg) constructor
- **Some widgets might only have non-default constructors available**
  - Widget designer may insist that client-provided state be specified via constructor arguments
  - You don't have control over third-party, so you can't provide a default constructor

Three Ways To Deal with Lack of Widget's Zero-Arg Constructor

- **Use @UiContructor inside the widget**
  - Requires least amount of coding
  - UI XML code becomes very clear
  - Downside: if 3rd party, must unwrap/code/rewrap JAR
- **Use @UiFactory inside the UiBinder owner widget**
  - Good choice if using a 3rd party widget library
  - No need to change the original source code
  - UI XML is still very clear
  - Allows for initialization with complex objects (not as attributes)
- **Use @UiField(provided=true) inside the UiBinder owner widget**
  - Same as @UiFactory, but UI XML is not as clear as *all* initialization happens in Java – no constructor attributes
  - Mostly used for sharing widget instances between views

@UiConstructor

- **Place @UiConstructor in front of widget constructor**
  - E.g., @UiConstructor public MyWidget(String name) {…}
- **Import widget library package as usual**
- **Inside .ui.xml, use constructor arg names as attributes**
  - *Names must match exactly* (unlike with setters)
  - E.g., …(String name){…} would be <c:Blah name=…>
- **Example preparation:**
  - Copied OptionalTextBox.java from GwtCustomWidgets1 project into GwtUiBinder1 project

Example: OptionalTextBox.java

```
package coreservlets.client.widget;
...
public class OptionalTextBox extends Composite {
  private TextBox textBox = new TextBox();
  private CheckBox checkBox = new CheckBox();
  private boolean checkedEnablesText;
  ...
  @UiConstructor
  public OptionalTextBox(String caption,
                         boolean checkedEnablesText,
                         boolean isChecked) {
    this.checkedEnablesText = checkedEnablesText;
    FlowPanel panel = new FlowPanel();
    initWidget(panel);
    setStyleName("core-opt-text-widget");
    textBox.setStyleName("core-opt-textbox");
    checkBox.setStyleName("core-opt-checkbox");
    ...
  }
```

UI XML must use caption, checkedEnablesText, and isChecked attributes.

Example: ContructorArg1.ui.xml

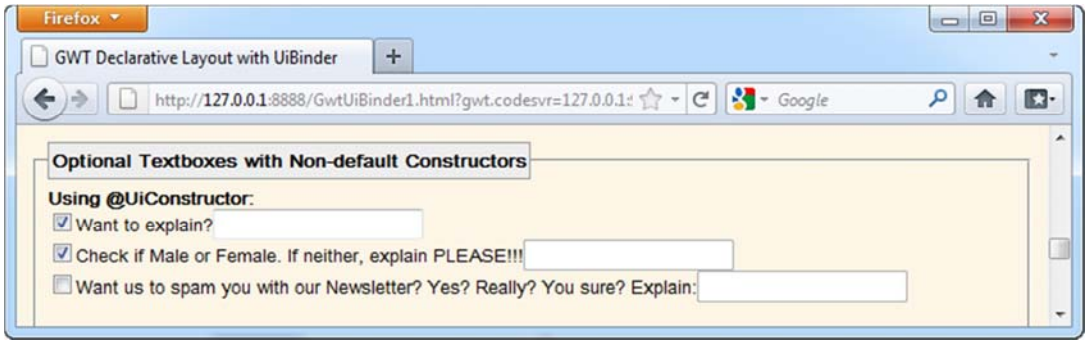
```
<!DOCTYPE ui:UiBinder SYSTEM
"http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
       xmlns:g="urn:import:com.google.gwt.user.client.ui"
       xmlns:core="urn:import:coreservlets.client.widget" …>
<g:HTMLPanel>
<div style="font-weight: bold; font-size: 12px;">
  Using @UiConstructor:</div>

<core:OptionalTextBox caption="Want to explain?"
                  checkedEnablesText="true" isChecked="true"/>
<core:OptionalTextBox caption="Check if Male or Female..."
                checkedEnablesText="false" isChecked="true"/>
<core:OptionalTextBox caption="Want us to spam ..."
                checkedEnablesText="true" isChecked="false"/>
<p/>
...
```

Example: GwtUiBinder1.html and Entry Class

```
...
<fieldset>
<legend>Optional Textboxes with Non-default
Constructors</legend>
<div id="optional-textbox-constructor"></div>
</fieldset>
<p/>
...
```

```
public class GwtUiBinder1 implements EntryPoint {
  public void onModuleLoad() {
    ...
    ConstructorArg1 constArg1 = new ConstructorArg1();
    RootPanel.get("optional-textbox-constructor")
                        .add(constArg1);
      ...
  }
}
```


Example: Result



@UiFactory

- **Create non-private method with widget type as the return type**
- **Place @UiFactory in front of that method**
  - E.g., @UiFactory MyWidget make(String name)
    {return new MyWidget(…);}
  - The method's argument names become attributes to the widget inside the UI XML as if they were part of a constructor
- **Import widget library package as usual**
- **Inside .ui.xml, use widget as usual**
  - Can still use attributes with setters besides the constructor args
- **Usually, this approach works well when using a 3rd party widget library**

Example Preparation

- **Use FancyOptionalTextBox from GwtCustomWidgets2 project**
  - Exported GwtCustomWidgets2 Java JAR into the GwtUiBinder1/war/WEB-INF/lib directory
    - Since this JAR has no server-side components, it can really be placed anywhere outside the war directory so it doesn't get deployed with the web app but only compiled into JavaScript
  - Adjusted GwtUiBinder1 project properties (Libraries) to include the gwtcustomwidgets2.jar
- **Inserted this line into GwtUiBinder1.gwt.xml:**
  <inherits name='coreservlets.GwtCustomWidgets2'/>

Example: ConstructorArg1.java

```
package coreservlets.client.ui;
...
public class ConstructorArg1 extends Composite {
  private static ConstructorArg1UiBinder uiBinder =
                 GWT.create(ConstructorArg1UiBinder.class);
  interface ConstructorArg1UiBinder
                 extends UiBinder<Widget, ConstructorArg1> {}
  ...
  @UiFactory
  FancyOptionalTextBox makeFancyTextBox(String caption,
                                        boolean checkedEnablesText,
                                        boolean isChecked) {
    return new FancyOptionalTextBox(caption,
                                    checkedEnablesText,
                                    isChecked);
  }
}
```

These names become attributes in XML UI file.

Example: ContructorArg1.ui.xml

```
<!DOCTYPE ui:UiBinder SYSTEM
"http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
       xmlns:g="urn:import:com.google.gwt.user.client.ui"
       xmlns:core="urn:import:coreservlets.client.widget"
       xmlns:fancy="urn:import:coreservlets.browser.widget">
<g:HTMLPanel>
...
<p/>
<div style="font-weight: bold; font-size: 12px;">
            Using @UiFactory:</div>
<fancy:FancyOptionalTextBox caption="Uncheck to clear
                nonsense comments you'll probably put here:"
                            checkedEnablesText="true"
                            isChecked="true"
                            clearTextOnDisable="true"/>
<p/>
...
```

Attributes corresponding to setters can still be used as usual. This one corresponds to setClearTextOnDisable(...).

Example: GwtUiBinder1.html and Entry Class

```
...
<fieldset>
<legend>Optional Textboxes with Non-default
Constructors</legend>
<div id="optional-textbox-constructor"></div>
</fieldset>
<p/>
...
```

Same as before since we are using the same UI XML file.

```
public class GwtUiBinder1 implements EntryPoint {
  public void onModuleLoad() {
    ...
    ConstructorArg1 constArg1 = new ConstructorArg1();
    RootPanel.get("optional-textbox-constructor")
                        .add(constArg1);
      ...
  }
}
```


Example: Result

@UiField(provided=true)

- **Inside .ui.xml, import widget library package**
- **Inside .ui.xml, use widget as usual without any constructor corresponding attributes**
  - Make sure to set the widget's ui:field="…" attribute
- **In corresponding Java class, declare field with @UiField(provided=true)**
  - This tells GWT not to call GWT.create(), up to you to wire it
  - Field must be instantiated before calling initWidget(…)
- **Usually, this approach is used with some factory that produces singleton widget**
  - Only one instance of it exists
  - This allows the entire UI to share the same instance of the widget
- **Disadvantage**
  - UI XML cannot use attributes corresponding to constructor args

Example Preparation

- **Use FancyOptionalTextBox from GwtCustomWidgets2 project**
- **Exported GwtCustomWidgets2 Java JAR into the GwtUiBinder1/war/WEB-INF/lib directory**
  - Since this JAR has no server-side components, it can really be placed anywhere outside the war directory so it doesn't get deployed with the web app but only compiled into JavaScript
  - Adjusted GwtUiBinder1 project properties (Libraries) to include the gwtcustomwidgets2.jar
- **Inserted this line into GwtUiBinder1.gwt.xml:**
  <inherits name='coreservlets.GwtCustomWidgets2'/>

Example: ConstructorArg1.java

```
...
public class ConstructorArg1 extends Composite {
  private static ConstructorArg1UiBinder uiBinder =
              GWT.create(ConstructorArg1UiBinder.class);
  interface ConstructorArg1UiBinder
              extends UiBinder<Widget, ConstructorArg1> {}

  @UiField(provided = true)
  FancyOptionalTextBox shared;

  public ConstructorArg1() {
    // You would usually retrieve this from a factory
    shared = new FancyOptionalTextBox("Sharing is caring!",
                                      true, false);
    initWidget(uiBinder.createAndBindUi(this));
  }
  ...
```

Important! This must be instantiated *before* calling initWidget(…).

Example: ContructorArg1.ui.xml

```
<!DOCTYPE ui:UiBinder SYSTEM
"http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
       xmlns:g="urn:import:com.google.gwt.user.client.ui"
       xmlns:core="urn:import:coreservlets.client.widget"
       xmlns:fancy="urn:import:coreservlets.browser.widget">
<g:HTMLPanel>
...
<p/>
<div style="font-weight: bold; font-size: 12px;">Using
       @UiField(provided=&amp;quot;true&amp;quot;):</div>
<fancy:FancyOptionalTextBox ui:field="shared"
                            clearTextOnDisable="true"/>
...
```

Attributes corresponding to setters can still be used as usual. This one corresponds to setClearTextOnDisable(…).

Example: GwtUiBinder1.html and Entry Class

```
...
<fieldset>
<legend>Optional Textboxes with Non-default
Constructors</legend>
<div id="optional-textbox-constructor"></div>
</fieldset>
<p/>
...
```

Same as before since we are using the same UI XML file.

```
public class GwtUiBinder1 implements EntryPoint {
  public void onModuleLoad() {
    ...
    ConstructorArg1 constArg1 = new ConstructorArg1();
    RootPanel.get("optional-textbox-constructor")
                        .add(constArg1);
      ...
  }
}
```


Example: Result

Special UiBinder Syntax for Panels

Overview

- **Idea**
  - Many Panel types have subelements that require special syntax
- **Most common issues**
  - Panels have sub-elements that cannot be bound to Java code
  - Panels may or may not allow raw HTML as body content of subelements

Example

- **Syntax for DockLayoutPanel**

```
<g:DockLayoutPanel unit='EM'>
  <g:north size='5'>
    <g:Label>Top</g:Label>
  </g:north>
  <g:center>
    <g:Label>Body</g:Label>
  </g:center>
  <g:west size='10'>
    <g:HTML>
      <ul>
        <li>Sidebar</li>
      </ul>
    </g:HTML>
  </g:west>
</g:DockLayoutPanel>
```

Unlike Label, HTML, etc., north, south, center, west, east do NOT represent runtime objects. They *can't* be bound.
(Being all lower case serves as a hint)

Not all widgets know what to do with raw HTML. Only those that implement HasHTML know how to handle HTML.

This is the reason we need to wrap our 'west' sidebar in an HTML widget that knows how to deal with raw HTML.

Working with CSS in UiBinder

Associating CSS Files: Choices

- **Host page**
  - Have to make sure GWT doesn't override your styles
  - Forces the browser to make another request
    - More on that when we discuss image loading
  - Docs say this is *deprecated* in GWT!
- **GWT Module XML file with <stylesheet>**
  - Old way, and GWT says it's now deprecated
- **Using CssResource inside ClientBundle**
  - Very efficient (more on that later)
  - Compile-time verification!
  - No more global CSS scope – goodbye CSS naming collisions!
- **Using inline <ui:style…> in UiBinder template**
  - Automatically creates a ClientBundle with CssResource

Example: InStyle1.ui.xml

```
<!DOCTYPE ui:UiBinder SYSTEM "http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
        xmlns:g="urn:import:com.google.gwt.user.client.ui">
<ui:style>
        .table {
                border: 2px blue solid;
        }
        .table tr+tr+tr {
                background: pink;
        }
        .randomNumber {
                font-style: italic;
                width: 250px;
        }
</ui:style>
<ui:style field="style2">
        .randomNumber {
                background-color: green;
                padding: 0px 4px 0px 4px;
                text-align: center;
                color: red;
                ...
        }
</ui:style>
```

All <ui:style> tags must be direct children of <ui:UiBinder>.

Both can be used within this page. The final name of the CSS class will be obfuscated to make sure these don't collide.

Prefix (i.e., namespace) for all styles contained in this style block.

If 'field' is omitted, the prefix defaults to 'style'. There can only be one default style block as above.

Example: InStyle1.ui.xml (cont)

```
<g:HTMLPanel>
<table class='{style.table}'>
  <tr><th>User Control</th>
      <th>Result</th></tr>
  <tr><td><g:Button ui:field="numberButton">Show Random Number</g:Button></td>
      <td><g:HTML ui:field="numberResult" addStyleNames="{style.randomNumber}">
           Num will go here</g:HTML></td></tr>
  <tr><td>N/A</td>
      <td><g:HTML addStyleNames="{style2.randomNumber}">Hurts your eyes,
           doesn't it?</g:HTML></td></tr>
</table><br/>
</g:HTMLPanel>
</ui:UiBinder>
```

CSS classes can be applied to regular HTML elements just like a regular CSS class. Multiple classes are separated by space.

Adds another class to the existing CSS classes assigned to this widget. Note the plural addStyleNames.

Example: InStyle1.java

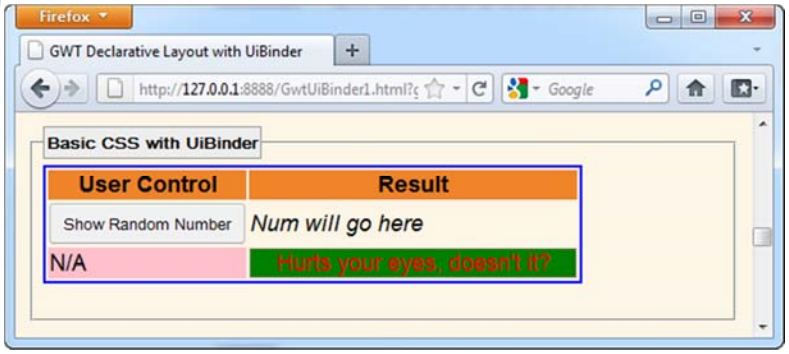
```
...
public class InStyle1 extends Composite {
  private static InStyle1UiBinder uiBinder =
                      GWT.create(InStyle1UiBinder.class);
  interface InStyle1UiBinder
                      extends UiBinder<Widget, InStyle1> {}

  @UiField Button numberButton;
  @UiField HTML numberResult;

  public InStyle1() {
    initWidget(uiBinder.createAndBindUi(this));
  }

  @UiHandler("numberButton")
  void displayRandomNumber(ClickEvent event) {
    numberResult.setText("Number: " + Math.random() * 10);
  }
}
```

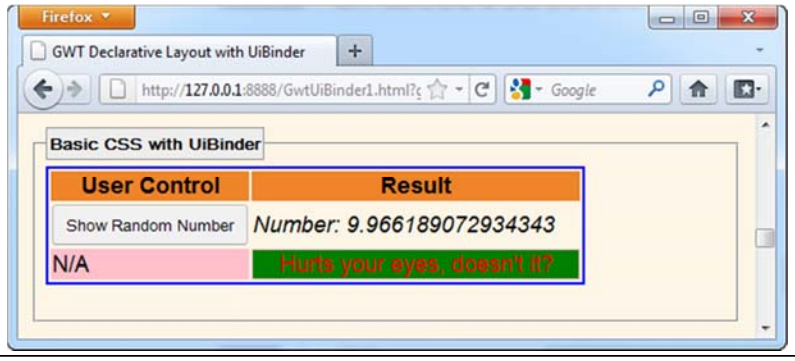
Example: GwtUiBinder.html and Entry Class

```
...
<fieldset>
<legend>Basic CSS with UiBinder</legend>
<div id="in-style1"></div>
</fieldset>
<p/>
...
```

```
public class GwtUiBinder1 implements EntryPoint {
  public void onModuleLoad() {
    ...
    InStyle1 inStyle1 = new InStyle1();
    RootPanel.get("in-style1").add(inStyle1);
       ...
  }
}
```


Example: Result





Example: InStyle2.ui.xml

```
<!DOCTYPE ui:UiBinder SYSTEM
"http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
       xmlns:g="urn:import:com.google.gwt.user.client.ui">
<ui:style field="mystyles" src="res/mystyles.css"/>
<ui:style field="dynamicStyle"
          type="coreservlets.client.ui.InStyle2.MyStyle">
       .moveRight {
              padding-left: 100px;
       }
       .moveLeft {
              padding-left: 2px;
       }
</ui:style>
<g:HTMLPanel>
<table><tr><th>User Control</th><th>Result</th></tr><tr><td>
<g:Button ui:field="diffButton" styleName="{mystyles.different}">
                              Different=Good</g:Button></td><td>
<g:HTML ui:field="notAlways">Not always!</g:HTML>
</td></tr>
</table><br/></g:HTMLPanel></ui:UiBinder>
```

Specifies a relative path to the CSS file.

Specifies that there will be a bound field inside InStyle2.java of type MyStyle that can access the classes defined in this style block.

This gives us the ability to dynamically assign inline styles to widgets in the template.

Example: InStyle2.java

```
...
public class InStyle2 extends Composite {
  private static InStyle2UiBinder uiBinder =
                  GWT.create(InStyle2UiBinder.class);
  interface InStyle2UiBinder
                  extends UiBinder<Widget, InStyle2> {}

  interface MyStyle extends CssResource {
    String moveRight();
    String moveLeft();
  }

  @UiField
  MyStyle dynamicStyle;

  ...
```

Must extend CssResource interface.

Method names must match the names of the classes inside the UI XML style block.

There is no requirement that all CSS classes within the style block be mapped here.

Example: InStyle2.java

```
  public InStyle2() {
    initWidget(uiBinder.createAndBindUi(this));
  }

  @UiHandler("diffButton")
  void moveNotAlways(ClickEvent event) {
    if (movedRight) {
      notAlways.removeStyleName(dynamicStyle.moveRight());
      notAlways.addStyleName(dynamicStyle.moveLeft());
      movedRight = false;
    } else {
      notAlways.removeStyleName(dynamicStyle.moveLeft());
      notAlways.addStyleName(dynamicStyle.moveRight());
      movedRight = true;
    }
  }
}
```


Example: res/mystyles.css

```
.different {
       width: 100px;
       height: 30px;
       border: 1px outset red;
       background: papayawhip;
}
```

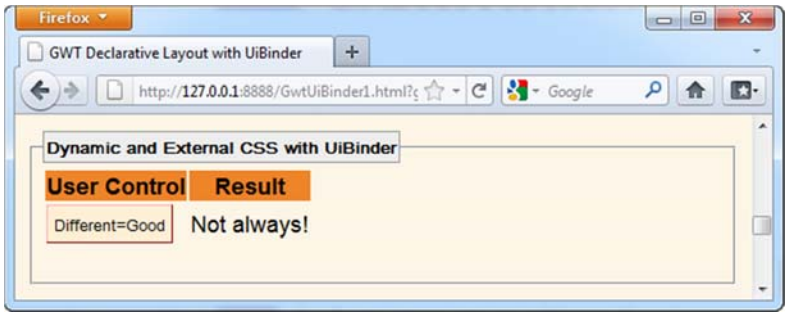
That's a real/valid color! Who knew?

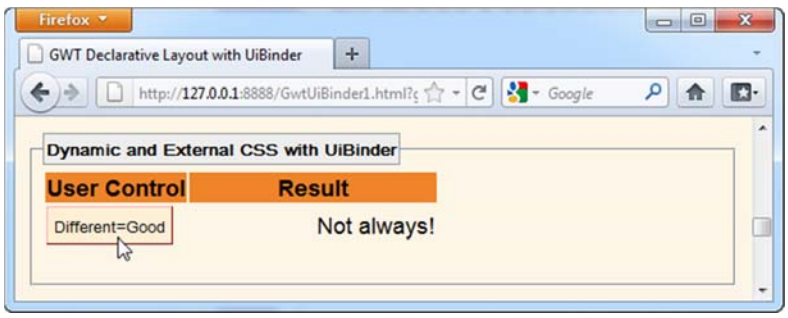
Example: GwtUiBinder.html and Entry Class

```
...
<fieldset>
<legend>Dynamic and External CSS with UiBinder</legend>
<div id="in-style2"></div>
</fieldset>
<p/>
...
```

```
public class GwtUiBinder1 implements EntryPoint {
  public void onModuleLoad() {
    ...
    InStyle2 inStyle2 = new InStyle2();
    RootPanel.get("in-style2").add(inStyle2);
       ...
  }
}
```


Example: Result





Working with Images in UiBinder

Problem

- **Large overhead**
  - Web apps mostly use *small* images for icons
  - HTTP request for each image – most traffic is overhead
- **Useless if-modified-since checks**
  - Even cached image needs to be checked if updated
    - Browser typically gets a 304 “Not Modified”
  - Images rarely change, so this is a waste
- **Limited HTTP connections**
  - HTTP 1.1 requires browser to use no more than 2 connections per domain/port
  - Large number of small images tie up the connections

Big Idea

- **ClientBundle solves the problem**
  - *Intelligently* stiches multiple images into one composite
  - Only 1 round trip to the server
- **Filename of the composite is a hash of file's contents**
  - Browser can safely cache it permanently
    - If anything changes, the file name will change as well
- **No more standard size place holders for images**
  - Avoids "bouncy page" affect as the size of the image is discovered when bundle is created – size can be explicitly set when images from bundles are used
- **Bundles can contain other types of resources**
  - CSS, text, PDF, etc.

Example: Resources.java

```
package coreservlets.client.ui.res;
...
public interface Resources extends ClientBundle {
  public static final Resources INSTANCE =
                                        GWT.create(Resources.class);

  @Source("red-ajax-loader.gif")
  ImageResource redLoader();
  @Source("gwtDuke.jpg")
  ImageResource gwtDuke();
  @Source("duke.gif")
  ImageResource duke();
  @Source("resStyles.css")
  Style style();

  public interface Style extends CssResource {
    String different();
    String loader();
    String gwtDuke();
    String duke();
  }...
```

Forces GWT to create concrete ClientBundle.

Relative path to the location of this interface.

Including a CSS file in the bundle. The style() method is the accessor for the CSS class wired methods in the Style interface.

Example: resStyles.css

```
.different {
        width: 180px;
        height: 30px;
        border: 1px outset red;
        background: papayawhip;
}
@sprite .loader {
        gwt-image: "redLoader";
}
@sprite .gwtDuke {
        gwt-image: "gwtDuke";
}
@sprite .duke {
        gwt-image: "duke";
}
```

Tells GWT to pull out a "clip" out of the composite image which is referenced "redLoader". The gwt-image is a special GWT property for this purpose.

The value of gwt-image must match the name of the method in Resources.

Example: InStyle3.ui.xml

```
<!DOCTYPE ui:UiBinder SYSTEM
"http://dl.google.com/gwt/DTD/xhtml.ent">
<ui:UiBinder xmlns:ui="urn:ui:com.google.gwt.uibinder"
        xmlns:g="urn:import:com.google.gwt.user.client.ui">
<ui:with field="res"
          type="coreservlets.client.ui.res.Resources"/>
<g:HTMLPanel><table>
  <tr><th>User Control</th><th>Result</th></tr><tr><td>
    <g:Button styleName="{res.style.different}">
                Resource Styled!</g:Button></td><td>
    <g:Image resource="{res.redLoader}"/></td></tr>
  <tr><td>GWT Duke</td><td>
    <g:HTML styleName="{res.style.gwtDuke}"/></td></tr>
  <tr><td>Regular Duke</td><td>
    <div class="{res.style.duke}"></div></td></tr>
  <tr><td><g:Button
            styleName="{res.style.different} {res.style.duke}"/>
      </td><td>
    <g:Image resource="{res.redLoader}"/></td></tr>
</table><br/></g:HTMLPanel></ui:UiBinder>
```

Example: InStyle3.java

```
...
public class InStyle3 extends Composite {
  private static InStyle3UiBinder uiBinder =
                 GWT.create(InStyle3UiBinder.class);
  interface InStyle3UiBinder
                 extends UiBinder<Widget, InStyle3> {}

  public InStyle3() {
    initWidget(uiBinder.createAndBindUi(this));
    Resources.INSTANCE.style().ensureInjected();
  }
}
```

Important! Often forgotten!

Make sure CSS resources get injected into the DOM of the browser.

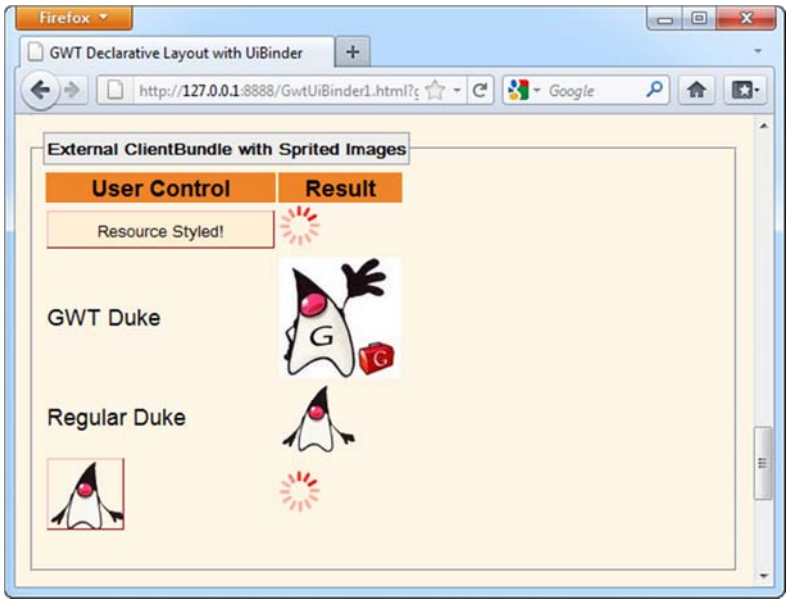
Calling this method multiple times has no effect, so it's OK to call it from multiple templates.

Example: GwtUiBinder.html and Entry Class

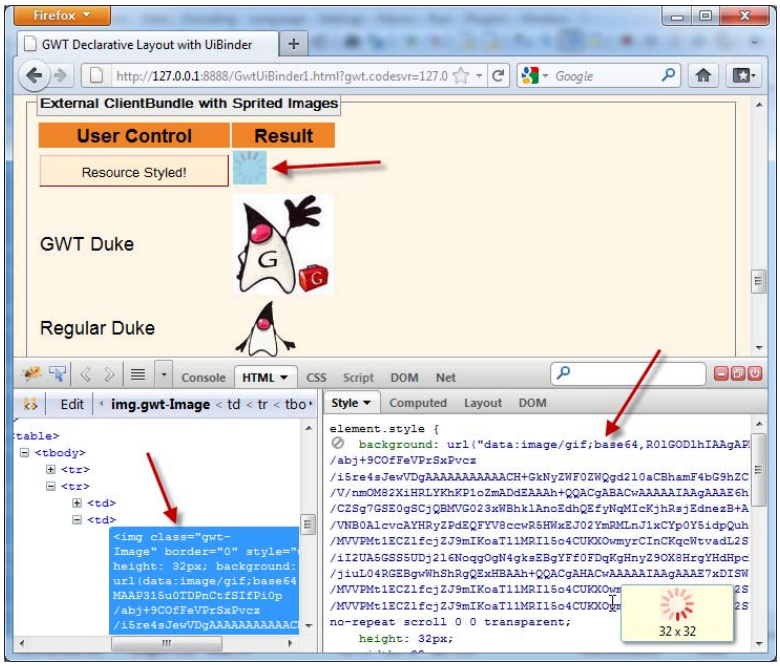
```
...
<fieldset>
<legend>External ClientBundle with Sprited Images</legend>
<div id="in-style3"></div>
</fieldset>
<p/>...
```

```
public class GwtUiBinder1 implements EntryPoint {
  public void onModuleLoad() {
    ...
    InStyle3 inStyle3 = new InStyle3();
    RootPanel.get("in-style3").add(inStyle3);
    ...
  }
}
```

Example: Result



Example: Result (Firebug)



Wrap-Up

Summary

- **Use XML attributes that correspond to widget's setter methods (like a JavaBean)**
  - Can pass primitives (or wrappers) or Strings
- **Use @UiConstructor if possible**
  - If *you* wrote the Widget, you can use @UiConstructor in Widget code. Then constructor can be used from XML.
- **Look at JavaDocs before using a panel**
  - To see if special syntax is required for UI XML
- **Use CSS with <ui:style…>**
  - Use its 'src' attribute to point to a central CSS file
- **Use ClientBundle approach**
  - To group images and thus improve HTTP performance

Questions?