The Google Web Toolkit (GWT): Creating Custom Widgets
(GWT 2.4 Version)

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

Customized Java EE Training: http://courses.coreservlets.com/
GWT, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Ajax and GWT training, please see courses at http://courses.coreservlets.com/.

Taught by the author of Core Servlets and JSP, More Servlets and JSP, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

Contact hall@coreservlets.com for details
Topics in This Section

- Motivation
- Three ways of building custom widgets
- Basic steps
- Example: OptionalTextBox
- Affecting layout of internal widgets
- Creating and using separate GWT modules
- Example: FancyOptionalTextBox
Motivation

- **Code reuse**
  - Applies to UI development just like any other part of your application
- **Easier maintenance**
  - Update one place if changes are required
- **Divvy up the work on the team without stepping on each other’s toes**
  - Usually self-contained enough so that it can developed relatively separately
- **Promotes thinking about UI functionality as set of loosely related components**
- **Create widget that GWT doesn’t provide**
  - Check if someone already has such a widget

Before Developing Your Own: Third-Party GWT Widget Libraries

- **GWT Widget Incubator**
  - Trial for widgets that may become standard.
    - http://code.google.com/p/google-web-toolkit-incubator/w/list
- **Ext-GWT (GXT)**
  - Large set of rich widgets from the makers of Ext-JS. Native implementations, not just JSNI wrappers.
- **Tatami**
  - Complete JSNI wrapper for Dojo. (only compatible with GWT 1.7 so far)
    - http://code.google.com/p/tatami/
- **Smart GWT**
  - GWT version of the extensive SmartClient library.
    - http://code.google.com/p/smartgwt/
- **GWT Component Library**
  - Small set of widgets plus complete Scriptaculous wrapper
    - http://gwt.components.googlepages.com/
Third-Party GWT Widget Libraries: What to Watch Out For

• Is library native GWT or just a JSNI wrapper?
  – If just a wrapper, consider using that library directly without GWT
  – Debugging can be very hard
  – Interaction with the library can be very different from regular GWT

• Is it an “all or nothing” library?
  – Some libraries force you to use their APIs for everything
    • Quite different from regular GWT
    • E.g., SmartGWT (it is also a thin JSNI wrapper)
  – You basically have to learn GWT-like library all over again

Third-Party GWT Widget Libraries: What to Watch Out For (cont)

• Even if not “all or nothing”, does it force you into a non-GWT way of interaction?
  – Can’t just plug it into your UI like you would a regular custom GWT widget

• Do they have active user support?
  – GWT discussion group has over 80 emails a day!

• Do they document how to use their widget well?
  – Even if they claim to document well
    • Take their more complex example and take a look at the JavaDoc APIs they used

• Button line: think of long-term cost
  – Sometimes using a bit less fancy widget or taking a bit longer upfront to develop your own is cheaper
Sometimes Management is Right

• Hard for developers to admit, but sometimes “long term cost” is not the highest priority

Three Ways of Building Custom Widgets

• Extend special Composite class
  – Most common and easiest way of creating custom widgets
  – Uses existing widgets to create a composite view with custom functionality
    • More complex widgets like TabPanel are built this way (TabBar & DeckPanel)

• Use lower level GWT DOM Java APIs
  – This is how most basic widgets are built
    • E.g., Button, TextBox, etc.
  – Quite complicated and only necessary in very specialized circumstances

• Use JavaScript and wrap it in a widget using JSNI
  – JSNI (JavaScript Native Interface) – See later tutorial
  – You would have to use deferred binding to isolate browser specific behavior
Basic Steps

• Create class that extends Composite
  – Class still needs to be in the JavaScript translatable package (.client by default)
• Call initWidget(Widget w) in its constructor
  – Tells GWT what widget to treat this new widget as
  – Usually, this is container widget (e.g., some container panel)
    • The rest of the internal widgets are attached to the container widget
• Optionally give new widget custom style name using setStyleName(String style)
  – Make sure to document this style name in JavaDoc
    • Be nice! Don’t make users dig through your code!

Basic Steps (continued)

• Create custom methods if necessary
  – Self-explanatory (i.e., custom presentation logic)
• Create public pass-through methods if necessary
  – Internal widgets are hidden from clients
  – However, some methods of those widgets might be directly needed by the client, so write a wrapper around those methods, passing the client-supplied values directly to that widget
  – If all of the methods of an internal widget are needed by the client, consider just extending that widget instead of extending Composite
    • The whole point of extending Composite is that you can hide the internals of inner widgets
• Use in your code like any other widget
Example: Optional Text Box

• **Goals:**
  – Demo basics of creating a custom widget
  – A widget that has a checkbox and textbox
  – Textbox gets disabled depending on the state of checkbox
    • Make this flexible: let client specify which one is “disabled” state, checked or unchecked
  – Expose internal widgets through documented CSS classes
  – Demo controlling look and even internal layout of custom widget by the client
    • Includes letting client control state-dependent look of internal widgets, e.g., what does disable textbox look like

• **Assumptions**
  – Already created new GWT projects, cleared out sample code, etc.

```java
package coreservlets.client.widget;
...
public class OptionalTextBox extends Composite {
    private TextBox textBox = new TextBox();
    private CheckBox checkBox = new CheckBox();
    private boolean checkedEnablesText;

    /** Style this overall widget with .core-opt-text-widget
     * CSS class.<br>
     * Style textbox with .core-opt-textbox CSS class.<br>
     * Style checkbox with .core-opt-checkbox CSS class<br>
     * Style disabled textbox with .core-opt-textbox-disabled
     * @param ... */
    public OptionalTextBox(String caption, boolean checkedEnablesText, boolean isChecked) {
        ...
    }
```

Must be in .client or a subpackage
Internal widgets hidden from clients
private boolean checkedEnablesText;
...

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");
    panel.add(checkBox);
    panel.add(textBox);
    checkBox.setText(caption);
    checkBox.setValue(isChecked);
    enableTextBox(checkedEnablesText, isChecked);
    checkBox.addClickHandler(new CheckBoxHandler());
}

private void enableTextBox(boolean checkedEnables, boolean isChecked) {
    boolean enable = (checkedEnables && isChecked)
                  || (!checkedEnables && !isChecked);
    textBox.setStyleDependentName("disabled", !enable);
    textBox.setEnabled(enable);
}

public void setCaption(String caption) {
    checkBox.setText(caption);
}

private class CheckBoxHandler implements ClickHandler {
    @Override
    public void onClick(ClickEvent event) {
        enableTextBox(checkedEnablesText, checkBox.getValue());
    }
}
Example: Host HTML page

```html
<!doctype html>
<html>
<head><title>GWT Custom Widgets</title>
<link rel="stylesheet"
      href="./css/styles.css"
      type="text/css"/>
<script type="text/javascript" language="javascript"
        src="gwtcustomwidgets1/gwtcustomwidgets1.nocache.js"></script>
</head>
<body>
...
<fieldset>
<legend>Optional Textbox 1</legend>
<div id="optional-textbox-1"></div>
</fieldset>
<p/>
<fieldset>
<legend>Optional Textbox 2</legend>
<div id="optional-textbox-2"></div>
</fieldset>
<p/>
<fieldset>
<legend>Optional Textbox 3</legend>
<div id="optional-textbox-3"></div>
</fieldset>
<p/>

Example: EntryPoint Class

```java
public class GwtCustomWidgets1 implements EntryPoint {

  public void onModuleLoad() {
    OptionalTextBox optTextBox1 =
      new OptionalTextBox("Want to explain?", true, true);
    RootPanel.get("optional-textbox-1").add(optTextBox1);

    OptionalTextBox optTextBox2 = new OptionalTextBox(
      "Check if Male or Female. If neither, "+ "explain PLEASE!!!", false, true);
    RootPanel.get("optional-textbox-2").add(optTextBox2);

    ...}
```
Example: style.css

```css
... .core-opt-text-widget .core-opt-textbox {
    margin-left: 5px;
    border: #3F9 solid 1px;
}
.core-opt-text-widget .core-opt-textbox-disabled {
    border: #CCC solid 1px;
}
#optional-textbox-1 .core-opt-text-widget,
#optional-textbox-2 .core-opt-text-widget,
#optional-textbox-3 .core-opt-text-widget {
    width: 530px;
}
...```

When enabled, show bright green border around the textbox.

When disabled, show grey border.

Ensure internal widgets don’t slide on top of another if browser window is made too small, i.e., always stay this minimum width.

Example: Result

Initial state

After unchecking the checkboxes
Affecting Layout of Internal Widgets

Styling Internal Widgets Example: EntryPoint Class

```java
public class GwtCustomWidgets1 implements EntryPoint {

    public void onModuleLoad() {
        OptionalTextBox optTextBox1 =
            new OptionalTextBox("Want to explain?", true, true);
        RootPanel.get("optional-textbox-1").add(optTextBox1);
        ...

        OptionalTextBox optTextBox3 = new OptionalTextBox("Want us to spam you with our Newsletter? Yes?\n+ " Really? You sure? Explain why below.",
            true, false);
        RootPanel.get("optional-textbox-3").add(optTextBox3);
        ...
    }
}
```

Create & add just like any regular widget.
Styling Internal Widgets Example:

```
... 
core-opt-text-widget .core-opt-textbox {
  margin-left: 5px;
  border: #3F9 solid 1px;
}
core-opt-text-widget .core-opt-textbox-disabled {
  border: #CCC solid 1px;
}
#optional-textbox-1 .core-opt-text-widget,
#optional-textbox-2 .core-opt-text-widget,
#optional-textbox-3 .core-opt-text-widget {
  width: 530px;
}
#optional-textbox-3 .core-opt-text-widget .core-opt-checkbox {
  margin-bottom: 3px;
  display: block;
}
```

24 TextBox is already a block-level element, but the way GWT implements CheckBox is through <span>, so need to force it to be a block-level element.

You can achieve a drastically different look of the custom widget through this type of technique.

Example: Result

Block-level elements automatically stack on top of each other inside FlowPanel (i.e., div element)
Creating and Using Separate GWT Modules

Motivation

- Modularizing your project is always good
  - Promotes what we all learned in CS101: *Unit of code should do one thing and do it well*
- Allows use of your unit of functionality to be reused in other modules & projects
  - Without extra unnecessary “baggage”
  - Perfect if you want create a GWT widget library and distribute it
- Team role separation
- Separate building and testing – faster compilation & development
Basic Steps

• Create new GWT Web Application Project
  – Do not let Eclipse generate project sample code

• Create new GWT Module
  – Module name can be anything, but usually same as name of the EclipseProject
  – Usually, no need to specify entry-point in Module.gwt.xml as this module will not be runnable
    • We’ll just include it in another (runnable) module

• Develop custom widget as usual in new module

Basic Steps (continued)

• Export your project as Java JAR
  – Make sure to include class files and source files
  – Exclude everything in the war directory as well as Eclipse artifacts like .classpath, .project

• Add newly created JAR file to the list of libraries of your other project
  – Or put it on the CLASSPATH somehow

• Tell your main GWT module to include GWT code from external library
  – Do this in ModuleName.gwt.xml using the <inherits> tag
  – Name of the module to include is fully qualified path to the IncludedModule.gwt.xml file without .gwt.xml

• Use newly included widget as usual
Example Overview

- **Create FancyOptionalTextBox**
  - Same as OptionalTextBox, plus:
    - On disabling the textbox, clear its contents
      - Helpful if reading the values later
    - Clearing the textbox should remember what was entered there before in case user changes their mind
    - Fix a major design deficiency in OptionalTextBox: no pass-through methods to retrieve values of internal CheckBox and TextBox
- **Package FancyOptionalTextBox in a separate GWT module**
  - We’ll use a separate eclipse project altogether for clearer separation
- **Use new widget in our original GWT project**

FancyOptionalTextBox Example: Create New GWT Web Project

Uncheck this to save yourself a lot of cleanup later.
FancyOptionalTextBox Example: Create New GWT Web Project

Right-click on GwtCustomWidgets2 project in Eclipse, choose New - Module

FancyOptionalTextBox Example: Create New GWT Web Project (con)
FancyOptionalTextBox Example: GwtCustomWidgets2.gwt.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "..." 
<module>
    <inherits name="com.google.gwt.user.User" />
    <source path="browser" />
</module>
```

Changed package that GWT should translate into JavaScript. In this module only classes under the .browser package will be translatable into JavaScript, not .client.

Includes basic GWT widgets and classes needed for development, i.e., include CheckBox, TextBox, etc.

FancyOptionalTextBox Example: FancyOptionalTextBox.java

```java
package coreservlets.browser.widget;
...
public class FancyOptionalTextBox extends Composite {
    ...
    private String previousText = "";
    ...
    public FancyOptionalTextBox(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");
        panel.add(checkBox);
        panel.add(textBox);
        checkBox.setText(caption);
        checkBox.setValue(isChecked);
        enableTextBox(checkedEnablesText, isChecked);
        checkBox.addClickHandler(new CheckBoxHandler());
    }
```

Constructor is the same as in OptionalTextBox

Translatable to JavaScript code now must reside inside .browser package.
private void enableTextBox(boolean checkedEnables, boolean isChecked) {
    boolean enable = (checkedEnables && isChecked)
        || (!checkedEnables && !isChecked);
    textBox.setStyleDependentName("disabled", !enable);
    clearOrRestoreTextBox(enable);
    textBox.setEnabled(enable);
}

private void clearOrRestoreTextBox(boolean enabledTextBox) {
    if (clearTextOnDisable && !enabledTextBox) {
        previousText = textBox.getText();
        textBox.setText("");
    } else {
        textBox.setText(previousText);
    }
}

public void setCaption(String caption) {
    checkBox.setText(caption);
}

public String getText() {
    return textBox.getText();
}

public boolean isChecked() {
    return checkBox.getValue();
}

public void setClearTextOnDisable(boolean clearTextOnDisable) {
    this.clearTextOnDisable = clearTextOnDisable;
}

private class CheckBoxHandler implements ClickHandler {
    @Override
    public void onClick(ClickEvent event) {
        enableTextBox(checkedEnablesText, checkBox.getValue());
    }
}
Since our widget doesn’t have any server-side components, we don’t really have to place it into the lib directory of the other project.

Uncheck ‘war’ folder and .classpath, .project

Refresh GwtCustomWidgets1 project (select it and press F5).

Right-click on the project name – Project Properties.

Click Add JARs… and choose gwtcustomwidgets2.jar
FancyOptionalTextBox Example: GwtCustomWidgets1.gwt.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='gwtcustomwidgets1'>
   <inherits name='com.google.gwt.user.User'/>
   <inherits name='com.google.gwt.user.theme.clean.Clean'/>
   <inherits name='coreservlets.GwtCustomWidgets2'/>

   <entry-point
      class='coreservlets.client.GwtCustomWidgets1'/>

   <source path='client'/>
</module>
```

Note that even though our newly included GWT library uses .browser package, we do NOT need to include this here. Since we inherited GwtCustomWidgets2 module, the .browser package is now automatically JavaScript translatable.

Note: it's a bad idea for this module to place code into .browser package without explicitly declaring it here.

---

FancyOptionalTextBox Example: GwtCustomWidgets1 EntryPoint

```java
package coreservlets.client;
...
public class GwtCustomWidgets1 implements EntryPoint {

   public void onModuleLoad() {
      ...
      OptionalTextBox optTextBox3 = new OptionalTextBox(
         "Want us to spam you with our Newsletter? Yes?"
         + " Really? You sure? Explain why below.",
         true, false);
      RootPanel.get("optional-textbox-3").add(optTextBox3);

      FancyOptionalTextBox fancyOptTextBox = new FancyOptionalTextBox(
         "Uncheck to clear nonsense comments you'll probably put here:",
         true, true);
      fancyOptTextBox.setClearTextOnDisable(true);
      RootPanel.get("optional-textbox-4").add(fancyOptTextBox);
   }
}
```
FancyOptionalTextBox Example: Result (Development Mode)

- Initially checked, we fill in some text.
- Then, uncheck the checkbox – clears and disables textbox.
- Then, check the checkbox again – enables and restores previously entered text.

FancyOptionalTextBox Example: Result (Hosted Mode)

GWT-compiled GwtCustomWidgets1 project
Wrap-Up

Summary

• **3rd party libraries: compare with care**
  – *Might* be easier just to develop what you need yourself

• **Extend Composite to create custom widget**
  – Call initWidget() (or setElement()) in constructor
  – Call setStyleName() to expose CSS class for styling
  – Provide public pass-through methods

• **Affect layout of internal widgets by exposing a documented CSS class and letting the client use CSS to style as they see fit**

• **Create separate GWT modules to achieve separation (faster compilation, testing, etc.)**
Questions?

JSF 2, PrimeFaces, Java 7, Ajax, jQuery, Hadoop, RESTful Web Services, Android, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training.

Customized Java EE Training: http://courses.coreservlets.com/
GWT, Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android. Developed and taught by well-known author and developer. At public venues or onsite at your location.