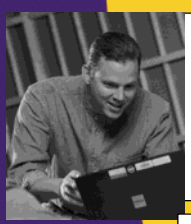
The Google Web Toolkit (GWT): Creating Custom Widgets

(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

For live Ajax & GWT training, see training 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.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, servlets/JSP, Ajax, jQuery, Android development, Java 7 or 8 programming, custom mix of topics
  - Courses available in any state or country. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, GWT, Hadoop, HTML5, RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

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

Overview

Motivation

- **Code reuse**
  - Build it once, use it many times
- **Easier maintenance**
  - Update one place if changes are required
- **Divvy up the work among team members**
  - Widgets are usually self-contained enough so that they can developed separately
- **Create new GUI widgets**
  - I.e., some visual effect not already provided by GWT.
    - But, you may want to check if it is already provided by a third-party GWT widget library

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.
  - Not free for commercial use (or free with source code release)
    - http://www.sencha.com/products/extgwt/
- **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

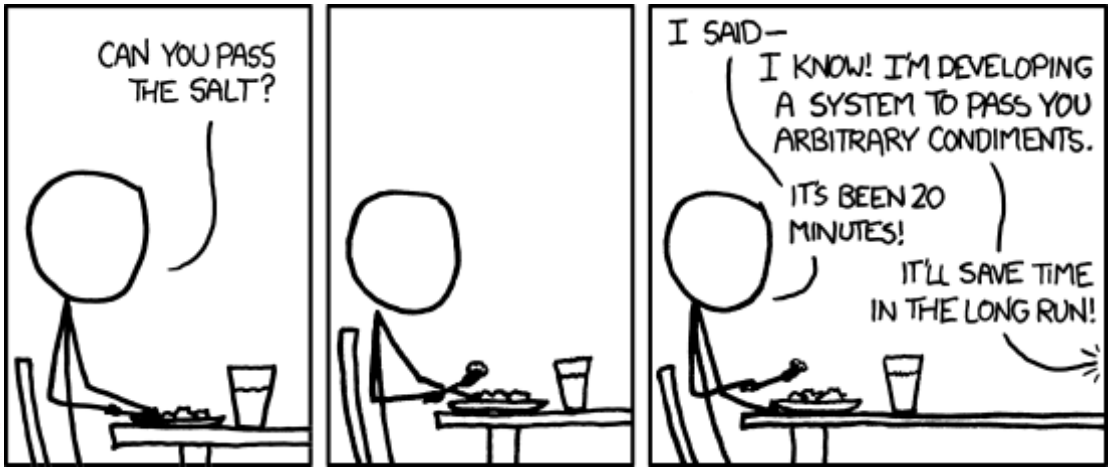
- **Is library native GWT, or is it just a JSNI wrapper?**
  - If just a wrapper, consider using that library directly without GWT
  - Debugging can be very hard
  - Interaction with the library can be very different from regular GWT
- **Is it an "all or nothing" library?**
  - Some libraries force you 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 you plug it into your application like you would a regular GWT widget?
- **Does library have an active user support?**
  - GWT discussion group has over 80 emails a day!
- **Is it well documented?**
  - Even if they claim to document well
    - Take their more complex example and take a look at the JavaDoc APIs they used
- **Bottom line: think of long-term cost**
  - Sometimes it is cheaper to use a bit less fancy widget or to take a bit longer up front to develop your own

Caution: Long-term Savings Not Always Better

- **Sometimes short-term costs are more important than long-term costs**



From xkcd.com and Randall Munroe

Three Ways of Building Custom Widgets

- **Extend 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 and 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
    - Possibly with deferred binding to isolate browser-specific behavior

Making Custom Widget by Extending Composite

Big Idea

- **Wrap multiple widgets in a container**
  - Usually a Panel
- **Give extra behavior**
  - Behavior that is more than the individual behaviors of the sub-widgets.
    - E.g., in example below, unchecking checkbox disables the textfield.
- **Make methods to pass settings through to underlying widgets**
  - So programmers can control look of the individual pieces
- **Example result**
  - End user sees a checkbox and textfield ☑ Want to explain?
  - GWT sees a FlowPanel ☑ Want to explain?

Basic Steps: Summary

1. **Extend Composite**
   - public class OptionalTextBox extends Composite
2. **Call initWidget(container) in constructor**
   - FlowPanel panel = new FlowPanel();
   - initWidget(panel);
3. **Assign CSS style names to new widget**
   - setStyleName("some-documented-css-name");
4. **Make methods to control new behavior**
   - Methods for high-level behavior of new widget
5. **Make pass-through methods**
   - To control look of individual pieces
6. **Use in your code like any other widget**

Basic Steps: Details

1. **Extend Composite**
   - public class OptionalTextBox extends Composite
     - Class still needs to be in the JavaScript-translatable package (.client by default)
     - If custom widget is a single underlying Widget with extra behavior, you can extend a lower-level Widget like TextBox instead of Composite
2. **Call initWidget(container) in constructor**
   - FlowPanel panel = new FlowPanel();
   - initWidget(panel);
     - Tells GWT what widget to treat this new widget as
     - Usually, this is container widget (e.g., some panel)
     - The rest of the internal widgets are attached to the container widget

Basic Steps: Details

3. Assign CSS style names to new widget

- Often to both the container (Panel) and the individual components.
- Be sure to clearly document these style names in the JavaDoc
  - Don't make other developers dig through your code
- Example
  - setStyleName("core-opt-text-widget");
  - textBox.setStyleName("core-opt-textbox");
  - checkBox.setStyleName("core-opt-checkbox");

Basic Steps: Details

4. Make methods to control new behavior

- Idea
  - The new widget will have some behavior that is more than just the individual behaviors of the component widgets.
- private methods (always)
  - You are almost sure to have private methods and event handlers that give behavior to your component.
  - E.g., in upcoming OptionalTextBox example, unchecking checkbox usually disables the textfield.
- public methods (sometimes)
  - In upcoming OptionalTextBox example, unchecking checkbox may or may not disable the textfield. In this case, we just pass argument controlling this to the constructor, but you could imagine a public method that lets programmer change this option later.

Basic Steps: Details

5. Make pass-through methods

- Programmer probably doesn't need to control all characteristics of the internal widgets, but probably needs to control some of them.
- E.g., in upcoming OptionalTextBox example, we let programmer change the caption later. So, we have public void setCaption(String caption) which just passes the caption to the setText method of the internal CheckBox.
- Alternative: make methods to retrieve the internal widgets (e.g., getCheckBox and getTextBox)
  - You usually want to hide the internal details, but if it is obvious that you use a CheckBox and TextBox, and you want to let developer control all options, this is possible approach. You are already exposing these details via the CSS classes anyhow.

Basic Steps: Details

6. Use in your code like any other widget

- HTML
  -
- Java
  - NewWidget yourNewWidget = new NewWidget(…);
  - RootPanel.get("placeholder").add(yourNewWidget);

Example: OptionalTextBox

Example: Optional Text Box

- **Behavior**
  - A widget that has a checkbox and textbox
  - Textbox gets disabled depending on the state of checkbox
    - Make this flexible: let client specify which is "disabled" state: checked or unchecked
  - Expose internal widgets through documented CSS classes
  - Illustrate 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 a disabled textbox looks like
- **Assumptions**
  - Already created new GWT projects, cleared out sample code, etc.

Step 1: Extend Composite (Also Document CSS classes)

```
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

Step 2: Call initWidget

```
  ...
  private boolean checkedEnablesText;
  ...
  public OptionalTextBox(String caption,
                         boolean checkedEnablesText,
                         boolean isChecked) {
    this.checkedEnablesText = checkedEnablesText;
    FlowPanel panel = new FlowPanel();
    initWidget(panel);
    ...
  }
```

Tells GWT to treat our new widget as if it were FlowPanel.

Step 3: Assign CSS style names

```
...
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);
  ...
}
...
```

Set style class names for entire widget. Equivalent to panel.setStyleName(...)

Expose styling of internal widgets to clients.

Step 4: Make Methods to Control New Behavior

```
...
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());
}
```

Step 4: Make Methods to Control New Behavior (Cont.)

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

  private class CheckBoxHandler implements ClickHandler {
    @Override
    public void onClick(ClickEvent event) {
      enableTextBox(checkedEnablesText, checkBox.getValue());
    }
  }
}
```

Make sure this is called initially after textBox.setStyle. Otherwise, if textbox is disabled at startup, the textbox will not *look* disabled.

Step 5: Make Pass-Through Methods

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

Step 6: Use Widget Normally (HTML Page)

```
...
<fieldset>
<legend>Optional Textbox 1</legend>
<div id="optional-textbox-1"></div>
</fieldset>
<br/>
<fieldset>
<legend>Optional Textbox 2</legend>
<div id="optional-textbox-2"></div>
</fieldset>
<br/>
...
```


Step 6: Use Widget Normally (EntryPoint Class)

```
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);
    ...
}
```

Create and add just like any widget.

Style Sheet

```
...
.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 green border around the textbox.

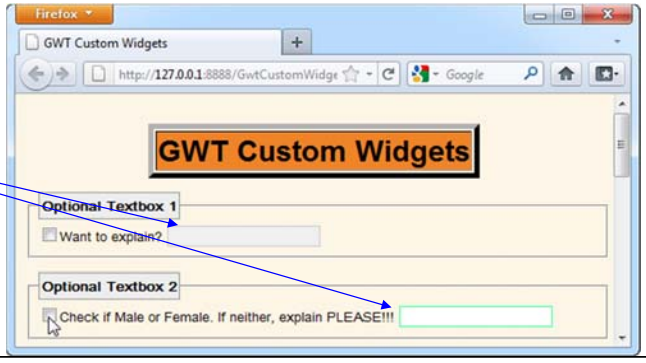
When disabled, display grey border.

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

Example: Result

Initial state

After unchecking the checkboxes



Using CSS to Control Layout of Internal Widgets

Example: Stacking CheckBox Above TextBox

- **Problem**
  - Checkbox (with label) for our OptionalTextBox is always on the same line as the textbox
  - May not be what we want if label contains a lot of text
- **General solution**
  - Make the checkbox with label stack on top of the textbox
- **Technique**
  - Use custom widget's exposed CSS classes to produce this layout

HTML Page

```
...
<fieldset>
<legend>Optional Textbox 1</legend>
<div id="optional-textbox-1"></div>
</fieldset>
<br/>
...
<fieldset>
<legend>Optional Textbox 3</legend>
<div id="optional-textbox-3"></div>
</fieldset>
<br/>
...
```


EntryPoint Class

```
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?"
           + " Really? You sure? Explain why below.",
           true, false);
    RootPanel.get("optional-textbox-3").add(optTextBox3);
    ...
  }
}
```

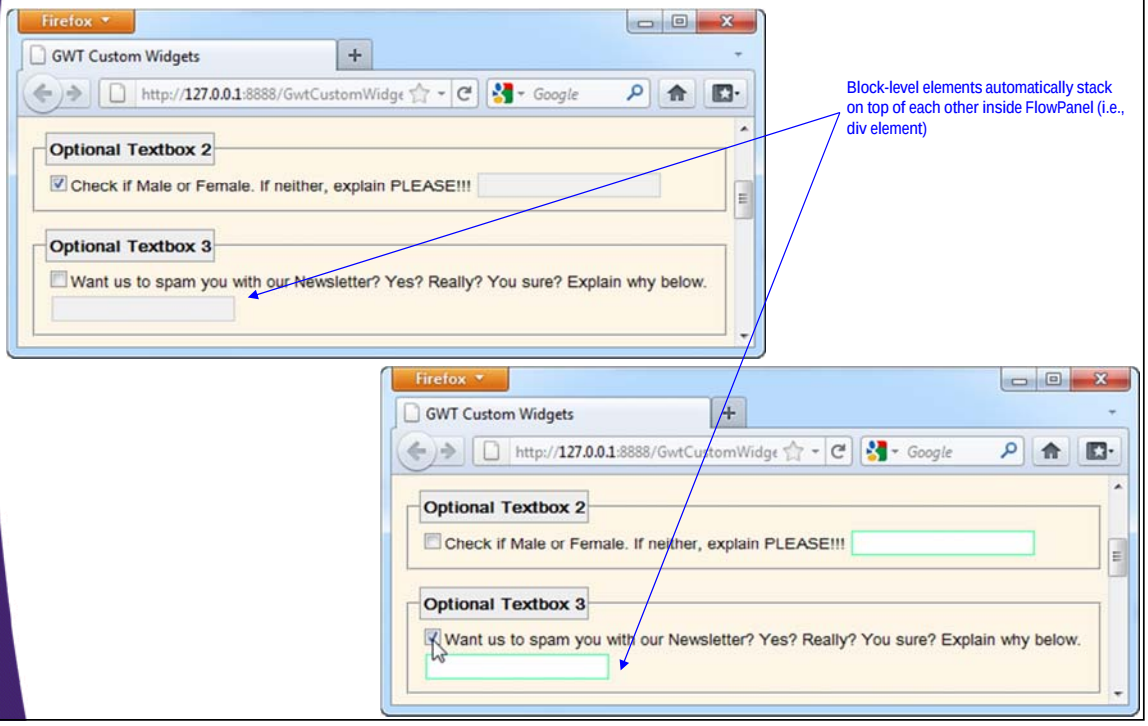
CSS File

```
...
#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;
}
```

TextBox is already a block-level element, but the way GWT implements CheckBox is through , 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



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 your functionality to be reused in other modules and projects**
  - Without unnecessary baggage
  - Ideal if you want create a GWT widget library and distribute it
- **Team role separation**
- **Separate building and testing**
  - Faster compilation and development

Basic Steps

1. **Create new GWT Web Application Project**
   - Do not let Eclipse generate project sample code
2. **Create new GWT Module**
   - Module name can be anything, but usually same as name of the Eclipse project
   - 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
3. **Develop custom widget as usual in new module**

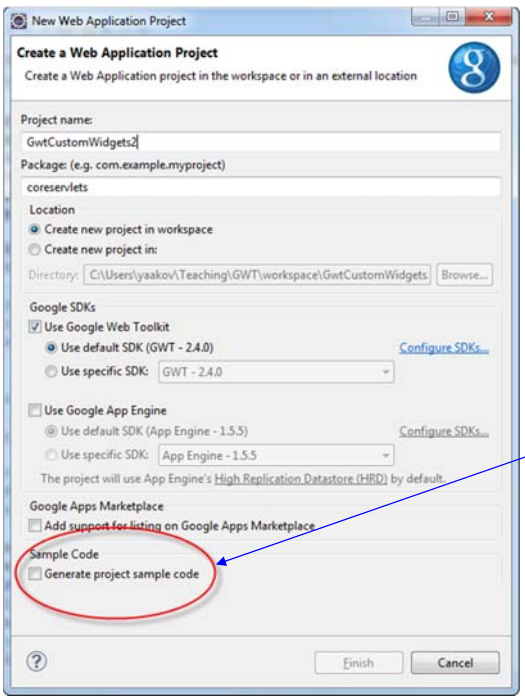
Basic Steps (Continued)

4. **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
5. **Add newly created JAR file to the list of libraries of your other project**
   - Or put it on the CLASSPATH somehow
6. **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*
7. **Use newly included widget as usual**

Example Overview

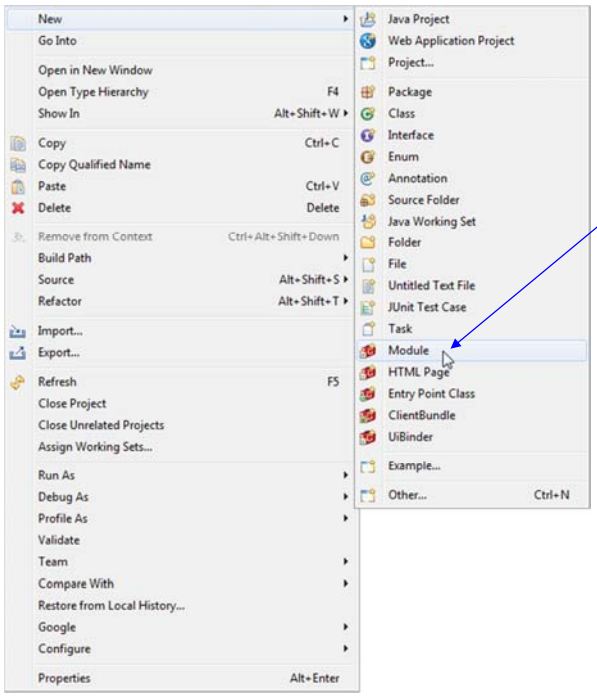
- **Create FancyOptionalTextBox**
  - Same as OptionalTextBox, plus:
  - On disabling the textbox, clear its contents
    - Helpful if reading the values later
  - Clearing the texbox should remember what was entered there (in case user changes their mind and re-enables)
  - Fix a major design deficiency 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



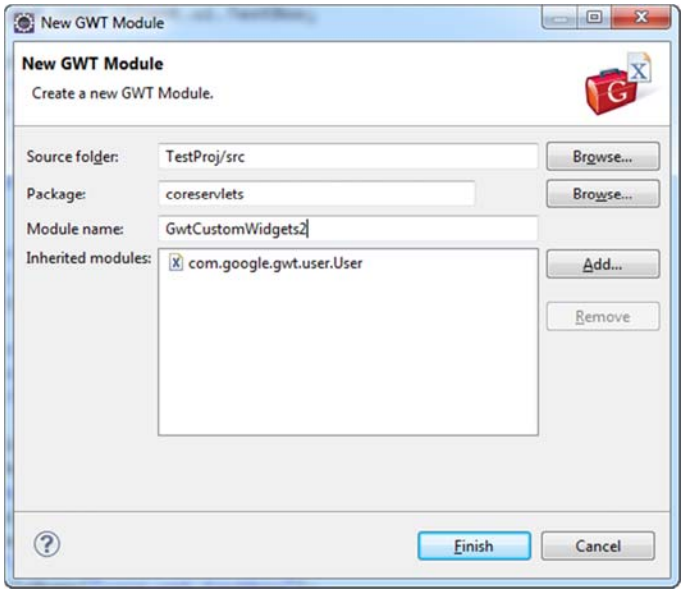
Uncheck this to save yourself a lot of cleanup later.

FancyOptionalTextBox Example: Create New GWT Web Project



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

FancyOptionalTextBox Example: Create New GWT Web Project (Cont.)



FancyOptionalTextBox Example: GwtCustomWidgets2.gwt.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 (not .client) will be translatable into JavaScript.

Includes basic GWT widgets and classes needed for development; e.g., CheckBox, TextBox, etc.

FancyOptionalTextBox Example: FancyOptionalTextBox.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());
  }
```

Code to be translated to JavaScript code now must reside inside .browser package.

Constructor is the same as in OptionalTextBox

FancyOptionalTextBox Example: FancyOptionalTextBox.java (con)

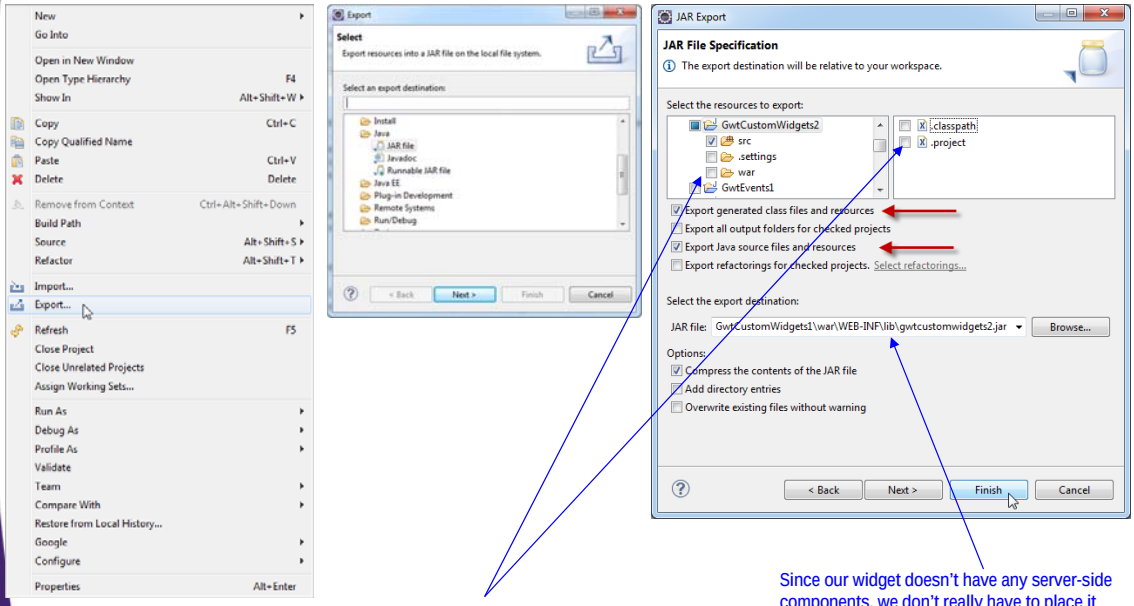
```
private void enableTextBox(boolean checkedEnables,
                           boolean isChecked) {
   boolean enable = (checkedEnables && isChecked)
       || (!checkedEnables && !isChecked);
   textBox.setStyleDependentName("disabled", !enable);
   clearOrRestoreTextBox(enable);
   textBox.setEnabled(enable);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

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

 ...
```


FancyOptionalTextBox Example: FancyOptionalTextBox.java (con)

```
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());
   } ...
```

Pass-through methods to enable clients to retrieve inner widget values.

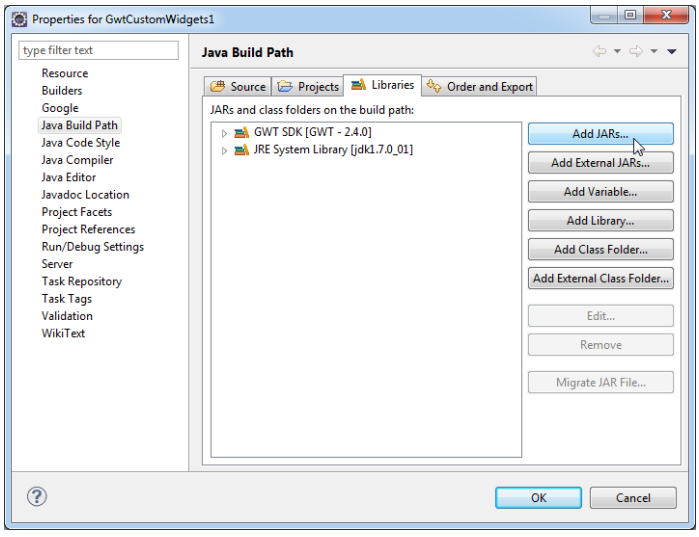
FancyOptionalTextBox Example: Export as Java JAR



Uncheck 'war' folder and .classpath, .project

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.

FancyOptionalTextBox Example: Import as Java JAR



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 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>
```

Fully qualified path to GwtCustomWidgets2.gwt.xml *without* the .gwt.xml part.

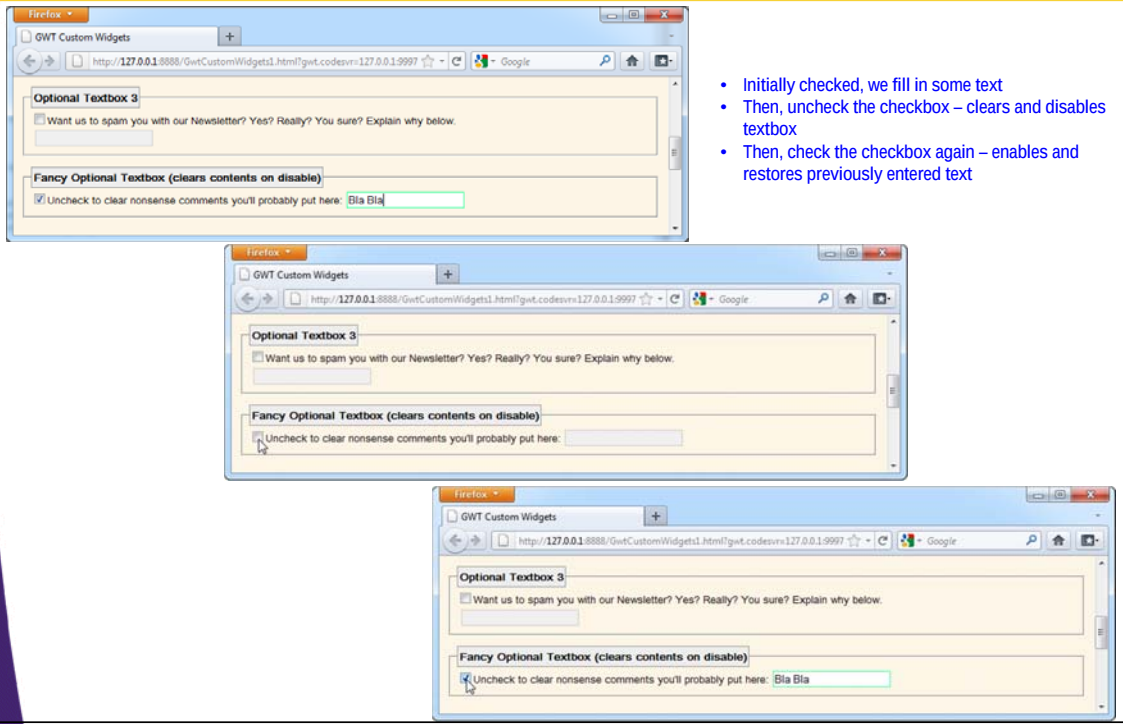
Note that even though our newly included GWT library uses .browser package, we do NOT need to include this here. Since we inherited GwtCustomWidgets2 module, the .browser package is now automatically JavaScript translatable.

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

FancyOptionalTextBox Example: GwtCustomWidgets1 EntryPoint

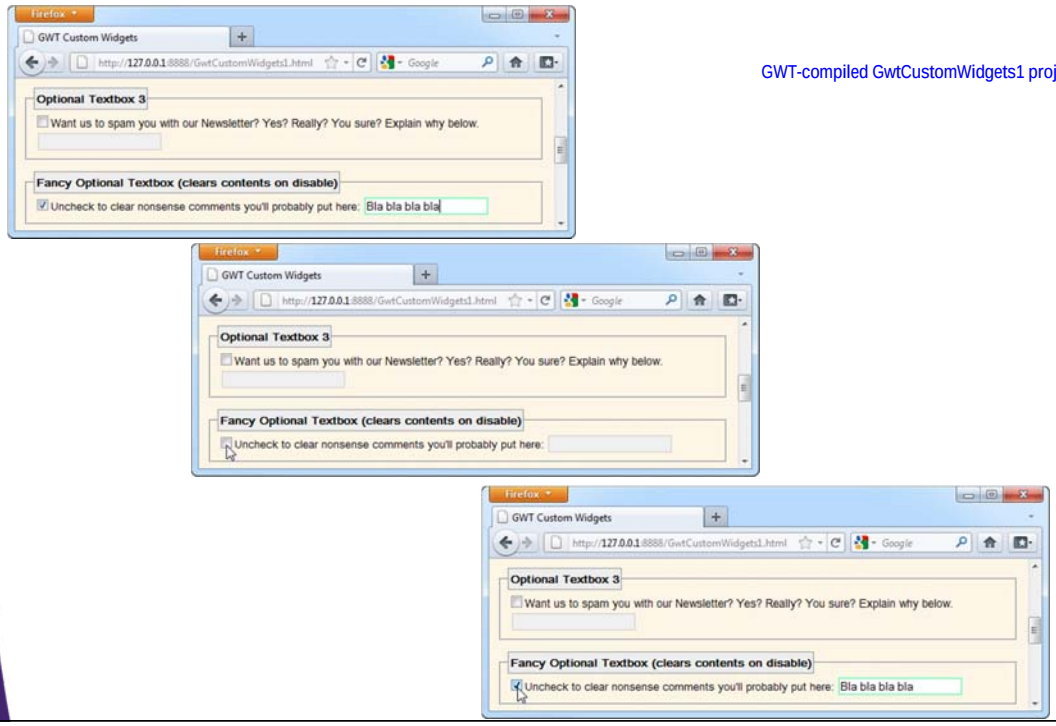
```
package coreservlets.client;
...
public class GwtCustomWidgets1 implements EntryPoint {

  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

- **First, consider using 3rd-party widget library**
  - But evaluate carefully, esp. if causes changes in GWT approach
- **When making your own components**
  1. Extend Composite
     - public class OptionalTextBox extends Composite
  2. Call initWidget(container) in constructor
     - FlowPanel panel = new FlowPanel();
     - initWidget(panel);
  3. Assign CSS style names to new widget
     - setStyleName("some-documented-css-name");
  4. Make methods to control new behavior
     - Methods for high-level behavior of new widget
  5. Make pass-through methods
     - To control look of individual pieces
  6. Use in your code like any other widget

Questions?