**Building User Interfaces**

GWT user interface classes are similar to those in existing UI frameworks such as [Swing](http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/package-summary.html) and [SWT](http://www.eclipse.org/swt/) except that the widgets are rendered using dynamically-created HTML rather than pixel-oriented graphics.

In traditional JavaScript programming, dynamic user interface creation is done by manipulating the browser's DOM. While GWT provides [access to the browser's DOM directly](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideAccessingDOM) using the [DOM class](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/DOM.html), it is far easier to use classes from the [Widget](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Widget.html) hierarchy. The Widget classes make it easier to quickly build interfaces that will work correctly on all browsers.

1. [**Cross-browser Support**](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideCrossBrowserSupport)
2. [**Understanding Layout**](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideUnderstandingLayout)
3. [**Widgets and Panels**](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideWidgetsAndPanels)
4. [**Creating Custom Widgets**](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideCreatingCustomWidgets)
5. [**Accessing the Browser's DOM**](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideAccessingDOM)
6. [**Events and Handlers**](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideEventsAndHandlers)
7. [**Applying Style**](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideStyleSheets)
8. [**Image Bundles**](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideImageBundles)

**Cross-browser Support**

GWT shields you from worrying too much about cross-browser incompatibilities. If you stick to built-in [widgets](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideWidgetsAndPanels) and [composites](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideCreatingCustomWidgets), your applications will work similarly on the most recent versions of Internet Explorer, Firefox, and Safari. (Opera, too, most of the time.) DHTML user interfaces are remarkably quirky, though, so make sure to test your applications thoroughly on every browser.

Whenever possible, GWT defers to browsers' native user interface elements. For example, GWT's [Button](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Button.html) widget is a true HTML <button> rather than a synthetic button-like widget built, say, from a <div>. That means that GWT buttons render appropriately in different browsers and on different client operating systems. We like the native browser controls because they're fast, accessible, and most familiar to users.

When it comes to styling web applications, [CSS](http://www.w3.org/Style/CSS/) is ideal. So, instead of attempting to encapsulate UI styling behind a wall of least-common-denominator APIs, GWT provides very few methods directly related to style. Rather, developers are encouraged to define styles in stylesheets that are linked to application code using [style names](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#setStyleName(java.lang.String)). In addition to cleanly separating style from application logic, this division of labor helps applications load and render more quickly, consume less memory, and even makes them easier to tweak during edit/debug cycles since there's no need to recompile for style tweaks.

**Tip:** If you find a need to implement a browser specific dependency, you can use a [JSNI](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideCodingBasics.html#DevGuideJavaScriptNativeInterface) method to retrieve the browser' UserAgent string.

  public static native String getUserAgent() /\*-{  
     return navigator.userAgent.toLowerCase();  
  }-\*/

**Understanding Layout**

Panels in GWT are much like their layout counterparts in other user interface libraries. The main difference lies in the fact that GWT panels use HTML elements such as DIV and TABLE to layout their child widgets.

**RootPanel**

The first panel you're likely to encounter is the [RootPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/RootPanel.html). This panel is always at the top of the containment hierarchy. The default RootPanel wraps the HTML document's body, and is obtained by calling [RootPanel.get()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/RootPanel.html#get()). If you need to get a root panel wrapping another element in the HTML document, you can do so using [RootPanel.get(String)](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/RootPanel.html#get(java.lang.String)).

**CellPanel**

[CellPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/CellPanel.html) is the abstract base class for [DockPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/DockPanel.html), [HorizontalPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/HorizontalPanel.html), and [VerticalPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/VerticalPanel.html). What these panels all have in common is that they position their child widgets within logical "cells". Thus, a child widget can be aligned within the cell that contains it, using [setCellHorizontalAlignment()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/CellPanel.html#setCellHorizontalAlignment(com.google.gwt.user.client.ui.Widget,%20com.google.gwt.user.client.ui.HasHorizontalAlignment.HorizontalAlignmentConstant)) and[setCellVerticalAlignment()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/CellPanel.html#setCellVerticalAlignment(com.google.gwt.user.client.ui.Widget,%20com.google.gwt.user.client.ui.HasVerticalAlignment.VerticalAlignmentConstant)). CellPanels also allow you to set the size of the cells themselves (relative to the panel as a whole) using [setCellWidth()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/CellPanel.html#setCellWidth(com.google.gwt.user.client.ui.Widget,%20java.lang.String)) and[setCellHeight()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/CellPanel.html#setCellHeight(com.google.gwt.user.client.ui.Widget,%20java.lang.String)).

Below is an example of using a [HorizontalPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/HorizontalPanel.html) from the Showcase sample application:

    // Create a Horizontal Panel  
    HorizontalPanel hPanel = new HorizontalPanel();  
  
    // Leave some room between the widgets  
    hPanel.setSpacing(5);   
  
    // Add some content to the panel  
    for (int i = 1; i < 5; i++) {  
      hPanel.add(new Button("Button " + i));  
    }

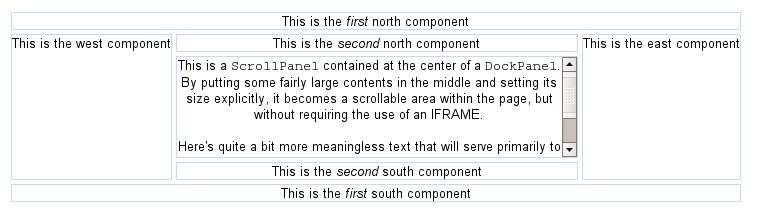
http://code.google.com/webtoolkit/doc/1.6/images/HorizontalPanel_Showcase.png

Each new Button widget is added to the right hand side of the panel, creating a horizontal row of widgets. The VerticalPanel works the same way, but adds new widgets to the bottom of the panel and lays them out vertically.

A more powerful layout mechanism is provided by the DockPanel. A DockPanel aligns its components using compass directions, where *north* points up on the screen. Below is an example of using a DockPanel from the Showcase sample application:

    DockPanel dock = new DockPanel();  
  
    // Allow 4 pixels of spacing between each cell  
    dock.setSpacing(4);  
  
    /\* Center each component horizontally within each cell  
     \* for each component added after this call.  
     \* A shortcut to calling dock.setCellHorizontalAlignment()  
     \* for each cell.  
     \*/  
    dock.setHorizontalAlignment(DockPanel.ALIGN\_CENTER);  
  
    // Add text widgets all around  
    dock.add(new HTML("This is the <i>first</i> north component"),  
        DockPanel.NORTH);  
    dock.add(new HTML("This is the <i>first</i> south component"),  
        DockPanel.SOUTH);  
    dock.add(new HTML("This is the east component"), DockPanel.EAST);  
    dock.add(new HTML("This is the west component"), DockPanel.WEST);  
    dock.add(new HTML("This is the <i>second</i> north component"),  
        DockPanel.NORTH);  
    dock.add(new HTML("This is the <i>second</i> south component"),  
        DockPanel.SOUTH);  
  
    // Add scrollable text in the center  
    HTML contents = new HTML("This is a <code>ScrollPanel</code> contained at "  
        + "the center of a <code>DockPanel</code>.  "  
        + "By putting some fairly large contents "  
        + "in the middle and setting its size explicitly, it becomes a "  
        + "scrollable area within the page, but without requiring the use of "  
        + "an IFRAME.<br><br>"  
        + "Here's quite a bit more meaningless text that will serve primarily "  
        + "to make this thing scroll off the bottom of its visible area.  "  
        + "Otherwise, you might have to make it really, really small in order "  
        + "to see the nifty scroll bars!");  
    ScrollPanel scroller = new ScrollPanel(contents);  
    scroller.setSize("400px", "100px");  
    dock.add(scroller, DockPanel.CENTER);

As you can see in the screenshot below, the order of adding components to the DocPanel is important if you add more than one component to the same direction. The first component added will be closest to the edge. The next component added will nest toward the inside of the panel.



**Tab Panel**

The [TabPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/TabPanel.html) displays a row of clickable tabs. Each tab is associated with a panel which can contain a sub panel or arbitrary HTML which is exposed in a main viewing area when the tab is selected. The main viewing area is implemented with a [DeckPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/DeckPanel.html). An example of using a Tab Panel follows:

    // Create a tab panel  
    TabPanel tabPanel = new TabPanel();  
  
    // Set the width to 400 pixels  
    tabPanel.setWidth("400px");  
  
    // Add a home tab  
    HTML homeText = new HTML("Click one the tabs to see more content.");  
    tabPanel.add(homeText, "Home");  
  
    // Add a tab with an image  
    VerticalPanel vPanel = new VerticalPanel();  
    vPanel.add(Showcase.images.gwtLogo().createImage());  
    tabPanel.add(vPanel, "GWT Logo");  
  
    // Add a third tab  
    HTML moreInfo = new HTML("Tabs are highly customizable using CSS");  
    tabPanel.add(moreInfo, "More Info");  
  
    // Make the first tab selected and the tab's content visible  
    tabPanel.selectTab(0);

The screenshot below shows the [TabPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/TabPanel.html) focused on the first tab. As you can see from the code example, you can add HTML or other panels to be displayed as the content of the tab. You might also notice that the widget looks different than the default Tab Panel. The Showcase demo uses [CSS Stylesheets](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideStyleSheets) to change the appearance of the panel.

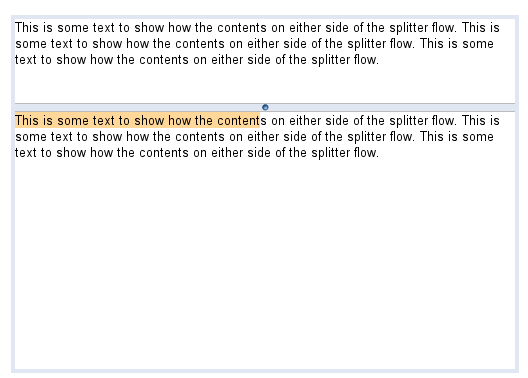


**Horizontal and Vertical Split panels**

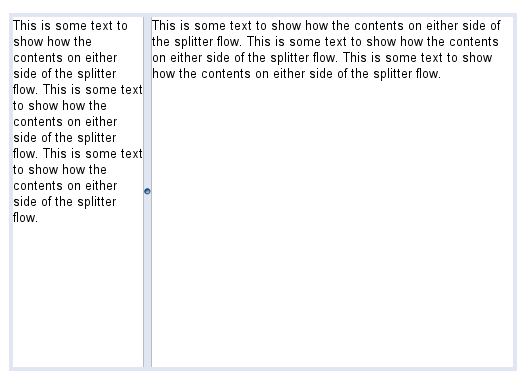
The [HorizontalSplitPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/HorizontalSplitPanel.html) and [VerticalSplitPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/VerticalSplitPanel.html) arrange two widgets in a single row or column and allow the user to interactively change the proportion of the width dedicated to each of the two widgets. Widgets contained within the split panels will be automatically decorated with scrollbars when necessary.

    // Create a Vertical Split Panel  
    VerticalSplitPanel vSplit = new VerticalSplitPanel();  
  
    // Set the bounding box in pixels  
    vSplit.setSize("500px", "350px");  
  
    /\* Set the position of the handle 30% from the top of the  
     \* panel.  
     \*/  
    vSplit.setSplitPosition("30%");  
  
    // Add some content  
    String randomText = "This is some text to show how the contents on either "  
        + "side of the splitter flow.   "  
        + "This is some text to show how the contents on either "  
        + "side of the splitter flow.   "  
        + "This is some text to show how the contents on either "  
        + "side of the splitter flow.   ";  
    vSplit.setTopWidget(new HTML(randomText));  
    vSplit.setBottomWidget(new HTML(randomText));

By manipulating the handle in the middle of the panel, the user can expose more of the upper or lower portion of the panel, as shown in the screenshot below.



The HorizontalSplitPanel works in the same way, only aligns its children horizontally.



**Other Panels**

Other panels include

|  |  |
| --- | --- |
| [FlowPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/FlowPanel.html) | A panel that formats its child widgets using the default HTML layout behavior. |
| [HTMLPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/HTMLPanel.html) | A panel that contains HTML, and which can attach child widgets to identified elements within that HTML. |
| [PopupPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/PopupPanel.html) | A panel that can "pop up" over other widgets. It overlays the browser's client area (and any previously-created popups). |
| [Grid](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Grid.html) | A rectangular grid that can contain text, html, or a child widget within its cells. It must be resized explicitly to the desired number of rows and columns. |
| [FlexTable](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/FlexTable.html) | A flexible table that creates cells on demand. It can be jagged (that is, each row can contain a different number of cells) and individual cells can be set to span multiple rows or columns. |
| [AbsolutePanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/AbsolutePanel.html) | An absolute panel positions all of its children absolutely, allowing them to overlap. |

See the Widget Gallery for screenshots or diagrams for these panels.

**Sizes and Measures**

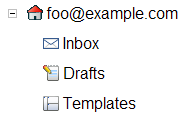
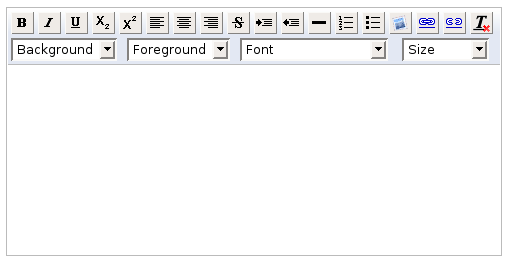
It is possible to set the size of a widget explicitly using [setWidth()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#setWidth(java.lang.String)), [setHeight()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#setHeight(java.lang.String)), and [setSize()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#setSize(java.lang.String,%20java.lang.String)). The arguments to these methods are strings, rather than integers, because they accept any valid CSS measurements, such as pixels (128px), centimeters (3cm), and percentage (100%).

**Widgets and Panels**

You construct user interfaces in GWT applications using [widgets](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Widget.html) that are contained within [panels](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Panel.html). Widgets allow you to interact with the user. Panels control the placement of user interface elements on the page. Widgets and panels work the same way on all browsers; by using them, you eliminate the need to write specialized code for each browser.

**Widgets**

Widgets define your applications input and output with the user. Examples of widgets include the following:

* [Button](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Button.html) A user clicks the mouse button to activate the button.
* http://code.google.com/webtoolkit/doc/1.6/images/Button.png
* [TextBox](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/TextBox.html) The application can display text and the user can type in the text box.
* http://code.google.com/webtoolkit/doc/1.6/images/TextBox.png
* [Tree](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Tree.html) A collapsible hierarchy of widgets.
* 
* [RichTextArea](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/RichTextArea.html) A text editor that allows R
* 

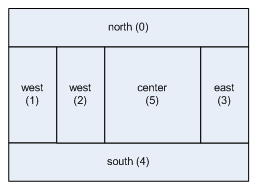
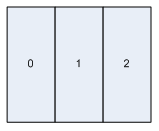
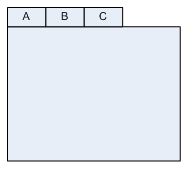
You are not limited to the set of widgets provided by GWT. There are a number of ways to [create custom widgets](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideCreatingCustomWidgets):

* You can bundle together existing widgets and create a *composite* widget.
* You can write GWT bindings to an existing JavaScript widget.
* You can create your own widget from scratch using either Java or JavaScript.

You can also use one or more of the many third party widget libraries written for GWT.

**Panels**

Panels contain widgets and other panels. They are used to define the [layout](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideUnderstandingLayout) of the user interface in the browser. Examples of panels include the following:

* [DockPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/DockPanel.html) Arranges widgets and subpanels using directional alignment.
* 
* [HorizontalPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/HorizontalPanel.html) Arranges widgets and subpanels horizontally.
* 
* [TabPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/TabPanel.html) Arranges subpanels and widgets under tabs.
* 
* [RootPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/RootPanel.html) This panel encapsulates your entire GWT user interface.

**Styles**

Visual styles are applied to widgets using [Cascading Style Sheets (CSS)](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideStyleSheets). Besides the default browser supplied definitions, each GWT widget and panel has pre-defined style sheet class definitions documented in the class reference documentation.

**See Also**

* Widget Gallery Diagrams and screen captures of the different GWT UI elements.
* [Creating Custom Widgets](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html" \l "DevGuideCreatingCustomWidgets) Discussion of how to create your own widgets in GWT.
* [Understanding Layout](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideUnderstandingLayout) Examples of how to use panels.

**Creating Custom Widgets**

GWT makes it easy to create custom user interface elements. There are three general strategies to follow:

* Create a widget that is a composite of existing widgets.
* Create an entirely new widget written in the Java language.
* Create a widget that wraps JavaScript using [JSNI](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideCodingBasics.html#DevGuideJavaScriptNativeInterface) methods.

There are numerous third party libraries that provide widgets you can integrate into your GWT module that were created using the strategies listed above.

**Building Composites**

The most effective way to create new widgets is to extend the [Composite](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Composite.html) class. A composite is a specialized widget that can contain another component (typically, a [panel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Panel.html)) but behaves as if it were its contained widget. You can easily combine groups of existing widgets into a composite that is itself a reusable widget. Some of the UI components provided in GWT are composites: for example, the [TabPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/TabPanel.html) (a composite of a TabBar and a DeckPanel) and the [SuggestBox](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/SuggestBox.html).

Rather than create complex widgets by subclassing [Panel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Panel.html) or another [Widget](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Widget.html) type, it's better to create a composite because a composite usually wants to control which methods are publicly accessible without exposing those methods that it would inherit from its Panel superclass.

**Example Composite Widget**

The following code snippet shows how to create a composite widget composed of a [TextBox](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/TextBox.html) widget and a [CheckBox](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/CheckBox.html) widget laid out in a [VerticalPanel](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/VerticalPanel.html).

package com.google.gwt.examples;  
  
import com.google.gwt.core.client.EntryPoint;  
import com.google.gwt.event.dom.client.ClickHandler;  
import com.google.gwt.user.client.ui.CheckBox;  
import com.google.gwt.user.client.ui.Composite;  
import com.google.gwt.user.client.ui.RootPanel;  
import com.google.gwt.user.client.ui.TextBox;  
import com.google.gwt.user.client.ui.VerticalPanel;  
import com.google.gwt.user.client.ui.Widget;  
  
public class CompositeExample implements EntryPoint {  
  
  /\*\*  
   \* A composite of a TextBox and a CheckBox that optionally enables it.  
   \*/  
  private static class OptionalTextBox extends Composite implements  
      ClickHandler {  
  
    private TextBox textBox = new TextBox();  
    private CheckBox checkBox = new CheckBox();  
  
    /\*\*  
     \* Constructs an OptionalTextBox with the given caption on the check.  
     \*   
     \* @param caption the caption to be displayed with the check box  
     \*/  
    public OptionalTextBox(String caption) {  
      // Place the check above the text box using a vertical panel.  
      VerticalPanel panel = new VerticalPanel();  
      panel.add(checkBox);  
      panel.add(textBox);  
  
      // Set the check box's caption, and check it by default.  
      checkBox.setText(caption);  
      checkBox.setChecked(true);  
      checkBox.addClickHandler(this);  
  
      // All composites must call initWidget() in their constructors.  
      initWidget(panel);  
  
      // Give the overall composite a style name.  
      setStyleName("example-OptionalCheckBox");  
    }  
  
    public void onClick(ClickEvent event) {  
      Object sender = event.getSource();  
      if (sender == checkBox) {  
        // When the check box is clicked, update the text box's enabled state.  
        textBox.setEnabled(checkBox.isChecked());  
      }  
    }  
  
    /\*\*  
     \* Sets the caption associated with the check box.  
     \*   
     \* @param caption the check box's caption  
     \*/  
    public void setCaption(String caption) {  
      // Note how we use the use composition of the contained widgets to provide  
      // only the methods that we want to.  
      checkBox.setText(caption);  
    }  
  
    /\*\*  
     \* Gets the caption associated with the check box.  
     \*   
     \* @return the check box's caption  
     \*/  
    public String getCaption() {  
      return checkBox.getText();  
    }  
  }  
  
  public void onModuleLoad() {  
    // Create an optional text box and add it to the root panel.  
    OptionalTextBox otb = new OptionalTextBox("Check this to enable me");  
    RootPanel.get().add(otb);  
  }  
}

**From Scratch in Java Code**

It is also possible to create a widget from scratch, although it is trickier since you have to write code at a lower level. Many of the basic widgets are written this way, such as [Button](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Button.html) and [TextBox](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/TextBox.html). Please refer to the implementations of these widgets to understand how to create your own.

To understand how to create your own, refer to the implementations of these widgets in the com.google.gwt.user.client.ui package. The source code is in gwt-user.jar.

**Using JavaScript**

When implementing a custom widget that derives directly from the [Widget](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Widget.html) base class, you may also write some of the widget's methods using JavaScript. This should generally only be done as a last resort, as it becomes necessary to consider the cross-browser implications of the native methods that you write, and also becomes more difficult to debug. For an example of this pattern in practice, see the [TextBox](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/TextBox.html) widget and the underlying JavaScript implementation of some of its methods in the TextBoxImpl class. You should use [deferred binding](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideCodingBasics.html#DevGuideDeferredBinding) to isolate browser specific code.

**Accessing the Browser's DOM**

Browsers provide an interface to examine and manipulate the on-screen elements using the [DOM](http://w3c.org/DOM/) (Document Object Model). Traditionally, JavaScript programmers use the DOM to program the user interface portion of their logic, and traditionally, they have had to account for the many differences in the implementation of the DOM on different browsers.

So that you don't have to worry (generally) about cross-browser support when implementing user interfaces, GWT provides a set of [widget and panel](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideWidgetsAndPanels)classes that wrap up this functionality. But sometimes you need to access the DOM. For example, if you want to:

* provide a feature in your user interface that GWT does not support
* write a new Widget class
* access an HTML element defined directly in the host page
* handle browser Events at a low level
* perform some filtering or other processing on an HTML document loaded into the browser

You could use [JSNI](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideCodingBasics.html#DevGuideJavaScriptNativeInterface) methods to access the browser's DOM, but GWT provides a convenience [DOM](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/DOM.html) class. The DOM class is not meant to be instantiated. Instead it is used as a collection of static methods that provides a way to walk and query the tree, plus a number of convenience routines. The DOM class also provides a layer of cross-browser support.

**Using the DOM to manipulate a widget**

Each widget and panel has an underlying DOM element that you can access with the [getElement()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#getElement()) method. You can use the getElement() method to get the underlying element from the DOM and manipulate or apply attributes to it with the DOM class.

The following example shows how to set a style attribute to change a widget's background color.

private HTML htmlWidget;  
  
// Other code to instantiate the widget...  
  
// Change the description background color.  
DOM.setStyleAttribute(htmlWidget.getElement(), "backgroundColor", "#fffe80");

Here, the getElement() method derived from the Widget superclass returns a DOM [Element](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/Element.html) object representing a node in the DOM tree structure and adds a style attribute to it.

This is an example where using the DOM isn't absolutely necessary. An alternative approach is to use [style sheets](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideStyleSheets) and associate different style classes to the widget using the [setStylePrimaryName()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#setStylePrimaryName(java.lang.String)) or [setStyleName()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#setStyleName(java.lang.String)) method instead.

**Finding an element in the DOM**

The following example shows how to combine a JSNI method with Java code to manipulate the DOM. First, we have a JSNI routine that will retrieve all the child elements that are Anchor tags. The element objects are assigned a unique ID for easy access from Java:

/\*\*  
 \* Find all child elements that are anchor tags,  
 \* assign a unique id to them, and return a list of  
 \* the unique ids to the caller.  
 \*/  
 private native void putElementLinkIDsInList(Element elt,  
                                                ArrayList list) /\*-{  
      var links = elt.getElementsByTagName("a");  
  
      for (var i = 0; i < links.length; i++ ) {  
        var link = links.item(i);  
        link.id = ("uid-a-" + i);  
        list.@java.util.ArrayList::add(Ljava/lang/Object;) (link.id);  
      }  
    }-\*/;

And what could you possibly do with a DOM element once you have found it? This code iterates through all the anchor tags returned from the above method and then rewrites where it points to:

/\*\*  
 \* Find all anchor tags and if any point outside the site,   
 \* redirect them to a "blocked" page.  
 \*/  
 private void rewriteLinksIterative() {  
   ArrayList links = new ArrayList();  
   putElementLinkIDsInList(this.getElement(), links);  
   for (int i = 0; i < links.size(); i++) {  
     Element elt = DOM.getElementById((String) links.get(i));  
     rewriteLink(elt, "www.example.com");  
   }  
 }  
  
/\*\*  
 \* Block all accesses out of the website that don't match 'sitename'  
 \* @param element An anchor link element  
 \* @param sitename name of the website to check.  e.g. "www.example.com"  
 \*/  
 private void rewriteLink(Element element, sitename) {  
   String href = DOM.getElementProperty(element, "href");  
  
   if (null == href) {  
       return;  
   }  
  
   // We want to re-write absolute URLs that go outside of this site  
   if (href.startsWith("http://") &&  
        !href.startsWith("http://"+sitename+"/") {  
       DOM.setElementProperty(element, "href",   
           "http://"+sitename+"/Blocked.html" );  
   }  
 }

The JSNI method set an ID on each element which we then used as an argument to [DOM.getElementById(id)](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/DOM.html#getElementById(java.lang.String)) to fetch the opaque Element handle in Java. Then the DOM static methods can be used to query and modify the properties of that element.

**Using the DOM to capture a browser event**

GWT contains an [Event](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/Event.html) class as an opaque handle to a native DOM Event. This object can be passed back and forth to JavaScript through [JSNI](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideCodingBasics.html#DevGuideJavaScriptNativeInterface) methods. It can also be manipulated by the GWT DOM class.

This example shows how to use the DOM methods to catch a keyboard event for particular elements and handle them before the [event](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideUserInterface.html#DevGuideEventsAndHandlers) gets dispatched:

  private ArrayList keyboardEventReceivers = new ArrayList();  
  
  /\*\*  
   \* Widgets can register their DOM element object if  
   \* they would like to be a trigger to intercept keyboard events  
   \*/  
  public void registerForKeyboardEvents (Element  e) {  
    this.keyboardEventReceivers.add(e);  
  }  
  
  /\*\*  
   \* Returns true if this is one of the keys we are interested in  
   \*/    
  public boolean isInterestingKeycode (int keycode) {  
     //...  
  }  
  
  /\*\*  
   \* Setup the event preview class when the module is loaded.  
   \*/  
  private void setupKeyboardShortcuts() {  
  
    // Define an inner class to handle the event  
    DOM.addEventPreview(new EventPreview() {  
      public boolean onEventPreview(Event event) {  
          
          Element elt = DOM.eventGetTarget(event);  
          int keycode = DOM.eventGetKeyCode(event);  
          boolean ctrl = DOM.eventGetCtrlKey(event);  
          boolean shift = DOM.eventGetShiftKey(event);  
          boolean alt = DOM.eventGetAltKey(event);  
          boolean meta = DOM.eventGetMetaKey(event);  
            
          if (DOM.eventGetType(event) != Event.ONKEYPRESS  
              || ctrl || shift || alt || meta  
              || keyboardEventReceivers.contains(elt)  
              || !isInterestingKeycode(keycode)) {  
            // Tell the event handler to continue processing this event.  
            return true;  
          }  
           
          GWT.log("Processing Keycode" + keycode, null );  
          handleKeycode(keycode);  
          // Tell the event handler that this event has been consumed  
          return false;  
      }});  
  }  
  
  /\*\*  
   \* Perform the keycode specific processing  
   \*/  
  private void handleKeycode (int keycode) {  
    switch (keycode) {  
        //...  
    }  
  }

**Events and Handlers**

Events in GWT use the *handler* model similar to other user interface frameworks. A handler interface defines one or more methods that the widget calls to announce an event. A class wishing to receive events of a particular type implements the associated handler interface and then passes a reference to itself to the widget to *subscribe* to a set of events. The [Button](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Button.html) class, for example, publishes click events. The associated handler interface is [ClickHandler](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/event/dom/client/ClickHandler.html).

The following example demonstrates how to add a custom ClickHandler subclass to an instance of a Button:

public void anonClickHandlerExample() {  
  Button b = new Button("Click Me");  
  b.addClickHandler(new ClickHandler() {  
    public void onClick(ClickEvent event) {  
      // handle the click event  
    }  
  });  
}

Using anonymous inner classes as in the above example can use excessive memory for a large number of widgets, since it results in the creation of many handler objects. Instead of creating separate instances of the ClickHandler object for each widget that needs to be listened to, a single handler can be shared between many widgets. Widgets declare themselves as the source of an event when they invoke a handler method, allowing a single handler to distinguish between multiple event publishers with an event object's getSource() method. This makes better use of memory but requires slightly more code, as shown in the following example:

public class HandlerExample extends Composite implements ClickHandler {  
  private FlowPanel fp = new FlowPanel();  
  private Button b1 = new Button("Button 1");  
  private Button b2 = new Button("Button 2");  
  
  public HandlerExample() {  
    initWidget(fp);  
    fp.add(b1);  
    fp.add(b2);  
    b1.addClickHandler(this);  
    b2.addClickHandler(this);  
  }  
  
  public void onClick(ClickEvent event) {  
    // note that in general, events can have sources that are not Widgets.  
    Widget sender = (Widget) event.getSource();  
  
    if (sender == b1) {  
      // handle b1 being clicked  
    } else if (sender == b2) {  
      // handle b2 being clicked  
    }  
  }  
}

**Applying Style**

GWT widgets rely on cascading style sheets (CSS) for visual styling.

In GWT, each class of widget has an associated style name that binds it to a CSS rule. Furthermore, you can assign an id to a particular component to create a CSS rule that applies just to that one component. By default, the class name for each component is gwt-<classname>. For example, the [Button widget](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Button.html) has a default style of gwt-Button.

In order to give all buttons a larger font, you could put the following rule in your application's CSS file:

.gwt-Button { font-size: 150%; }

All of the widgets created with the GWT toolkit will have a default class name, but a widget's style name can be set using [setStyleName()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#setStyleName(java.lang.String)). Static elements can have their class set in the HTML source code for your application.

Another way to use style sheets is to refer to a single widget. For that, you would need to know the value of the id attribute for the widget or DOM element.

By default, neither the browser nor GWT creates default id attributes for widgets. You must explicitly create an id for the elements you want to refer to in this manner, and you must insure that each "id" value is unique. A common way to do this is to set them on static elements in your [HTML host page](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideOrganizingProjects.html#DevGuideHostPage)

  <div id="my-button-id"/>

To set the id for a GWT widget, retrieve its DOM Element and then set the id attribute as follows:

  Button b = new Button();  
  DOM.setElementAttribute(b.getElement(), "id", "my-button-id")

This would allow you to reference a specific widget in a style sheet as follows:

#my-button-id { font-size: 100%; }

**Complex Styles**

Some widgets have multiple styles associated with them. [MenuBar](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/MenuBar.html), for example, has the following styles:

.gwt-MenuBar {

/\* properties applying to the menu bar itself \*/

}

.gwt-MenuBar .gwt-MenuItem {

/\* properties applying to the menu bar's menu items \*/

}

.gwt-MenuBar .gwt-MenuItem-selected {

/\* properties applying to the menu bar's selected menu items \*/

}

In the above style sheet code, there are two style rules that apply to menu items. The first applies to all menu items (both selected and unselected), while the second (with the -selected suffix) applies only to selected menu items. A selected menu item's style name will be set to "gwt-MenuItem gwt-MenuItem-selected", specifying that both style rules will be applied. The most common way of doing this is to use [setStyleName](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#setStyleName(java.lang.String)) to set the base style name, then [addStyleName()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#addStyleName(java.lang.String)) and [removeStyleName()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/UIObject.html#removeStyleName(java.lang.String)) to add and remove the second style name.

**Associating CSS Files**

There are two approaches for associating CSS files with your module:

* Using a <link> tag in the [host HTML page](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideOrganizingProjects.html#DevGuideHostPage).
* Using the <sytlesheet> element in the [module XML file](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideOrganizingProjects.html#DevGuideModuleXml).

You should use one approach or the other, but not both.

**Including Style sheets in the HTML Host Page**

Typically, style sheets are placed in a package that is part of your module's [public path](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideOrganizingProjects.html#DevGuideModules). All you need to do to reference them is simply include a <link> to the style sheet in your [host page](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideOrganizingProjects.html#DevGuideHostPage), such as:

  <link rel="stylesheet" href="mystyles.css" type="text/css"/>

**Including Style sheets in the Module XML file**

Another way to include your style sheet within your module is to use the <stylesheet> element in your [module XML file](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideOrganizingProjects.html#DevGuideModuleXml). This uses [automatic resource inclusion](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideOrganizingProjects.html#DevGuideAutomaticResourceIclusionn) to bundle the .css file with your module.

The difference between using a <link> tag in HTML and the <stylesheet> element in your module XML file is that with the mdoule XML file approach, the style sheet will always follow your module, no matter which host HTML page you deploy it from.

Why does this matter? Because if you create and share a module, it does not include a host page and therefore, you cannot guarantee the style sheet's availability. Automatic Resource Inclusion solves this problem. If you do not care about sharing or re-using your module then you can just use the standard HTML link rel stuff in the host page.

**Tip:** Use a unique name for the .css file with included resources to avoid collisions. If you automatically include "styles.css" and share your module and someone puts it on a page that already has "styles.css" there will be problems.

**GWT Visual Themes**

GWT comes with three default visual themes that you can choose from: standard, chrome, and dark. The standard theme uses subtle shades of blue to create an lively user interface. The chrome theme uses gray scale backgrounds for a refined, professional look. The dark theme uses dark shades of gray and black with bright indigo highlights for a bold, eye catching experience. When you inherit a visual theme, almost all widgets will have some default styles associated with them. The visual themes allow you to focus more time on application development and less time on styling your application.

By default, new GWT applications use the standard theme, but you can select any one of the themes mentioned above. Open your module XML file (gwt.xml) and uncomment the line that inherits the theme of your choice.

<!-- Inherit the default GWT style sheet. You can change       -->  
<!-- the theme of your GWT application by uncommenting          -->  
<!-- any one of the following lines.                           -->  
<!-- <inherits name='com.google.gwt.user.theme.standard.Standard'/> -->  
<!-- <inherits name="com.google.gwt.user.theme.chrome.Chrome"/> -->  
<inherits name="com.google.gwt.user.theme.dark.Dark"/>

GWT visual themes also come in RTL (right-to-left) versions if you are designing a website for a language that is written right-to-left, such as Arabic. You can include the RTL version by adding RTL to the end of the module name:

<inherits name="com.google.gwt.user.theme.dark.DarkRTL"/>

**Bandwidth Sensitive Applications**

If you are program a bandwidth sensitive application, such as a phone application, you may not want to require that users download the entire style sheet associated with your favorite theme (about 27k). Alternatively, you can create your own stripped down version of the style sheet that only defines the styles applicable to your application. To do this, first include the public resources associated with one of the themes by adding the following line to your gwt.xmlfile:

<inherits name='com.google.gwt.user.theme.standard.StandardResources'/>

Each theme has a "Resources" version that only includes the public resources associated with the theme, but does not inject a style sheet into the page. You will need to create a new style sheet and inject it into the page as described in the sections above.

Finally, copy the contents of the file public/gwt/standard/standard.css style sheet located in the package com.google.gwt.user.theme.standardinto your new style sheet. Strip out any styles you do not want to include, reducing the size of the file. When you run your application, GWT will inject your stripped down version of the style sheet, but you can still reference the files associate with the standard visual theme.

**Documentation**

It is standard practice to document the relevant CSS style names for each widget class as part of its documentation comment. For a simple example, see[Button](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Button.html). For a more complex example, see [MenuBar](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/MenuBar.html).

**Image Bundles**

An *image bundle* is a construct used to improve application performance by reducing the number of round trip HTTP requests to the server to fetch images. GWT can package many image files into a single large file to be downloaded from the server and managed as a Java object.

**ImageBundles Make Using Images More Efficient**

Typically, an application uses many small images for icons. In HTML, each image is stored in a separate file and the browser is asked to download each file from the web server as a separate HTTP transaction. This standard way of dealing with images can be wasteful in several ways:

* **Large overhead:** In a standard web application, an HTTP request has to be sent to the server for each image. In many cases, those images are icons and the actual image size is very small. In that case, the size of the image is often smaller than the HTTP response header that is sent back with the image data. That means that most of the traffic is overhead and very little of it actual content.
* **Useless freshness checks:** Traditional image handling is wasteful in other ways too. Even when the images have been cached by the client, a 304 ("Not Modified") request is still sent to check and see if the image has changed. Since images change infrequently, these freshness checks are also wasteful.
* **Blocking HTTP connections:** Furthermore, HTTP 1.1 requires browsers to limit the number of outgoing HTTP connections to two per domain/port. A multitude of image requests will tie up the browser's available connections, which blocks the application's RPC requests. In most applications, RPC requests are the real work that the application needs to do.

The end result of sending out many separate requests and freshness checks is slow application startup.

The GWT *image bundle* solves these problems. An image bundle is a composition of many images into a single image, along with an interface for accessing the individual images from within the composite. Users can define an image bundle that contains the images used by their application, and GWT will automatically create the composite image and provide an implementation of the interface for accessing each individual image. Instead of a round trip to the server for each image, only one round trip to the server for the composite image is needed.

Because the filename of the composite image is based on a hash of the file's contents, the filename will change only if the composite image is changed. This means that it is safe for clients to cache the composite image permanently, which avoids the unnecessary freshness checks for unchanged images. To make this work, the server configuration needs to specify that composite images never expire. In addition to speeding up startup, image bundles prevent the "bouncy" effect of image loading in browsers. While images are loading, browsers put a standard placeholder for each image in the UI. The placeholder is a standard size because the browser does not know what the size of an image is until it has been fully downloaded from the server. The result is a 'bouncy' effect, where images 'pop' into the UI once they are downloaded. With image bundles, the size of each individual image within the bundle is discovered when the bundle is created, so the size of the image can be explicitly set whenever images from a bundle are used in an application.

See the [ImageBundle API documentation](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/ImageBundle.html) for important information regarding:

* A potential security issue with the generation of the composite image on certain versions of the JVM
* Caching recommendations for image bundle files
* Protecting image bundle files with web application security constraints
* Using image bundles with the HTTPS protocol

**Creating and Using an Image Bundle**

To define an image bundle, the user needs to extend the [ImageBundle](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/ImageBundle.html) interface. The ImageBundle interface is a tag interface that can be extended to define new image bundles. The derived interface can have zero or more methods, where each method must have the following characteristics:

* The method takes no parameters
* The method has a a return type of [AbstractImagePrototype](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/AbstractImagePrototype.html)
* The method may have an optional gwt.resource metadata tag which specifies the name of the image file in the module's classpath

Valid image file types are png, gif, and jpg. If the image name contains '/' characters, it is assumed to be the name of a resource on the classpath, formatted as would be expected by ClassLoader.getResource(String). Otherwise, the image must be located in the same package as the user-defined image bundle.

If the gwt.resource metadata tag is not specified, then the following assumptions are made:

* The image filename is assumed to match the method name
* The extension is assumed to be either .png, .gif, or .jpg
* The file is assumed to be in the same package as the derived interface

In general, you should not create create the same image filename with different types. In the event that there are multiple image files with the same names but different extensions, the order of extension precedence is (1) png, (2) gif, then (3) jpg. This means that if you add foo to the ImageBundle and have files foo.png and foo.jpg, only foo.png will be picked up - foo.jpg will be ignored.

An image bundle for icons in a word processor application could be defined as follows:

public interface WordProcessorImageBundle extends ImageBundle {  
  
  /\*\*  
   \* Would match the file 'new\_file\_icon.png', 'new\_file\_icon.gif', or  
   \* 'new\_file\_icon.png' located in the same package as this type.  
   \*/  
  public AbstractImagePrototype new\_file\_icon();  
  
  /\*\*  
   \* Would match the file 'open\_file\_icon.gif' located in the same  
   \* package as this type.  
   \*   
   \* @gwt.resource open\_file\_icon.gif  
   \*/  
  public AbstractImagePrototype openFileIcon();  
  
  /\*\*  
   \* Would match the file 'savefile.gif' located in the package  
   \* 'com.mycompany.mygwtapp.icons', provided that this package is part  
   \* of the module's classpath.  
   \*   
   \* @gwt.resource com/mycompany/mygwtapp/icons/savefile.gif  
   \*/  
  public AbstractImagePrototype saveFileIcon();  
}

Methods in an image bundle return [AbstractImagePrototype](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/AbstractImagePrototype.html) objects rather than [Image](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/Image.html) objects, as you might have expected. This is becauseAbstractImagePrototype objects provide additional lightweight representations of an image. For example, the [AbstractImagePrototype.getHTML()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/AbstractImagePrototype.html#getHTML()) method provides an HTML fragment representing an image without having to create an actual instance of the Image widget. In some cases, it can be more efficient to manage images using these HTML fragments.

Another use of AbstractImagePrototype is to use [AbstractImagePrototype.applyTo(Image)](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/AbstractImagePrototype.html#applyTo(com.google.gwt.user.client.ui.Image)) to transform an existing Image into one that matches the prototype without having to instantiate another Image object. This can be useful if your application has an image that needs to be swapped depending on some user-initiated action.

Not all situations can be satisfied with a lightweight HTML fragment or by copying into an existing Image. For example, you may need to access a member of your Image bundle and pass an Image object into a setWidget() method to display your image in a TreeItem. In those cases, the[AbstractImagePrototype.createImage()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/user/client/ui/AbstractImagePrototype.html#createImage()) method can be used to generate new Image instances.

The following example shows how to use the image bundle that we just defined in your application. An Image object is needed in this case because theHorizontalPanel performs layout only on subclasses of Widget.

public void useImageBundle() {  
  WordProcessorImageBundle wpImageBundle = (WordProcessorImageBundle) GWT.create(WordProcessorImageBundle.class);  
  HorizontalPanel tbPanel = new HorizontalPanel();  
  tbPanel.add(wpImageBundle.new\_file\_icon().createImage());  
  tbPanel.add(wpImageBundle.openFileIcon().createImage());  
  tbPanel.add(wpImageBundle.saveFileIcon().createImage());  
}

**Tip:** Image bundles are immutable, so you can keep a reference to a singleton instance of an image bundle instead of creating a new instance every time the image bundle is needed.

**Image Bundles and Localization**

Sometimes applications need different images depending on the locale that the user is in. When using image bundles, this means that we need different image bundles for different locales.

Although image bundles and localization are orthogonal concepts, they can work together by having locale-specific factories create instances of image bundles. The best way to explain this technique is with an example. Suppose that we define the following ImageBundle for use by a mail application:

public interface MailImageBundle extends ImageBundle {  
  
  /\*\*  
   \* The default 'Compose New Message' icon if no locale-specific  
   \* image is specified.  
   \*  
   \* @gwt.resource compose\_new\_message\_icon.gif  
   \*/  
  public AbstractImagePrototype composeNewMessageIcon();  
  
  /\*\*  
   \* The default 'Help' icon if no locale-specific image is specified.  
   \* Will match 'help\_icon.png', 'help\_icon.gif', or 'help\_icon.jpg' in  
   \* the same package as this type.  
   \*/  
  public AbstractImagePrototype help\_icon();  
}

Suppose the application has to handle both English and French users. In that case, we will have to define these locale values in the [module XML file](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideOrganizingProjects.html#DevGuideModuleXml):

<module>  
  
    <!-- Inherit the core Web Toolkit stuff.      -->  
    <inherits name='com.google.gwt.user.User' />  
    <inherits name='com.google.gwt.i18n.I18N' />  
  
    <!-- For Localizable                          -->  
        <extend-property name='locale' values='fr' />  
        <extend-property name='locale' values='en' />  
  
    <!-- Other settings ...                       -->  
  
    
</module>

We define English and French variations of each image in MailImageBundle by creating locale-specific image bundles that extend MailImageBundle:

public interface MailImageBundle\_en extends MailImageBundle {  
  
  /\*\*  
   \* The English version of the 'Compose New Message' icon.  
   \* Since we are not overriding the help\_icon() method, this bundle  
   \* uses the inherited method from MailImageBundle.  
   \*  
   \* @gwt.resource compose\_new\_message\_icon\_en.gif  
   \*/  
  public AbstractImagePrototype composeNewMessageIcon();  
  
  
  /\* Note: No override for the help icon \*/  
  
}

public interface MailImageBundle\_fr extends MailImageBundle {  
  
  /\*\*  
   \* The French version of the 'Compose New Message' icon.  
   \*  
   \* @gwt.resource compose\_new\_message\_icon\_fr.gif  
   \*/  
  public AbstractImagePrototype composeNewMessageIcon();  
  
  /\*\*  
   \* The French version of the 'Help' icon.  
   \*  
   \* @gwt.resource help\_icon\_fr.gif  
   \*/  
  public AbstractImagePrototype help\_icon();  
}

The next step is to create a mechanism for choosing the correct image bundle based on the user's locale. By extending [Localizable](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/i18n/client/Localizable.html), we can create a locale-sensitive factory that will instantiate MailImageBundle with deferred binding.

public interface MailImageBundleFactory extends Localizable {  
  
  public MailImageBundle createImageBundle();  
}

The factory methods are instantiating the ImageBundle superclass using deferred binding. Invoking the [GWT.create()](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/core/client/GWT.html#create(java.lang.Class) method will cause a generator to create Java code to implement the details for the image bundle class for each factory:

public class MailImageBundleFactory\_en implements MailImageBundleFactory {  
  
  public MailImageBundle createImageBundle() {  
    return (MailImageBundle) GWT.create(MailImageBundle\_en.class);  
  }  
}

public class MailImageBundleFactory\_fr implements MailImageBundleFactory {  
  
  public MailImageBundle createImageBundle() {  
    return (MailImageBundle) GWT.create(MailImageBundle\_fr.class);  
  }  
}

Finally, we need to create a Factory to return the default implementation of our ImageBundle. Do this by naming the class with the same prefix as the interface, but ending with an underscore character:

public class MailImageBundleFactory\_ implements MailImageBundleFactory {  
  
  public MailImageBundle createImageBundle() {  
    return (MailImageBundle) GWT.create(MailImageBundle.class);  
  }  
}

Application code that utilizes a locale-sensitive image bundle might look something like this:

public void useLocalizedImageBundle() {  
  // Create a locale-sensitive MailImageBundleFactory  
  MailImageBundleFactory mailImageBundleFactory = (MailImageBundleFactory) GWT  
      .create(MailImageBundleFactory.class);  
  
  // This will return a locale-sensitive MailImageBundle, since we are using  
  // a locale-sensitive factory to create it.  
  MailImageBundle mailImageBundle = mailImageBundleFactory.createImageBundle();  
  
  // Get the image prototype for the icon that we are interested in.  
  AbstractImagePrototype helpIconProto = mailImageBundle.help\_icon();  
  
  // Create an Image object from the prototype and add it to a panel.  
  HorizontalPanel panel = new HorizontalPanel();  
  panel.add(helpIconProto.createImage());  
}

Creating the factory class also takes advantage of GWT's [deferred binding](http://code.google.com/intl/es-ES/webtoolkit/doc/1.6/DevGuideCodingBasics.html#DevGuideDeferredBinding) feature, this time based on the Localizable base class. The[GWT.create(Class)](http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/com/google/gwt/core/client/GWT.html#create(java.lang.Class)) method instructs the compiler to choose one of the several locale specific subclasses of MailImageBundleFactory available.

**Why did this Example use Deferred Binding Twice?**

If you are familiar with deferred binding, you might be thinking, *Why didn't we just make MailImageBundle extend the Localizable class directly?*

The answer is rooted in the fact that you cannot extend more than one class in Java. Both ImageBundle and the Localizable class are needed to implement this feature and they both use deferred binding. Hence the use of the *Factory method pattern*.