
3.8. Styling components

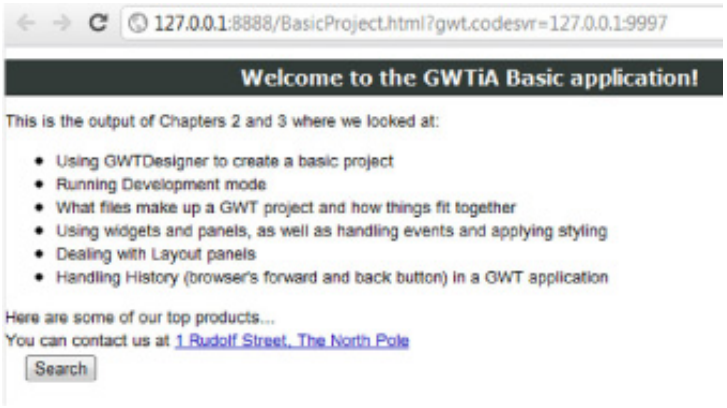
You can style widgets in the following five ways:

- Programmatically, such as saying `button.setWidth("100px");` in your code
- Low level via the DOM, by writing in your code `button.getElement().getStyle().setWidth(100);`
- By using Cascading Style Sheets, which means placing a definition in your CSS file `.gwt-button{width:100px;}`
- By inheriting one of GWT'S built-in themes in your module definition
- By using an expression language

The latter point on expression language we'll leave until chapter 6's discussion on GWT's UiBinder (declarative user interface design). We'll use the rest of this chapter to look at the other.

If you had no styling then the `BasicProject` application would look as uninspiring as figure 3.8. (You'll probably also see this view for a few seconds when first running in development mode because GWT is busy preparing itself but has already served the HTML.)

Figure 3.8. The unstyled chapter 3 `BasicProject` example looking less than appealing—but you'll fix that with some CSS styling.



It's not difficult to see that figure 3.8 is a long way from our mock-up. But you can fix it with some simple styling, starting with the first of our techniques.

3.8.1. Programmatic styling

All widgets can be styled using methods they inherit from `UIObject` (such as `setWidth` and `setSize`). To set the size of `HTMLPanels` and the `TabLayoutPanel` within `BasicProject`'s tab panel, you use the `setHeight` method, for example:

```
home.setHeight("400px");
```

You can look at the `styleTabPanelUsingUIObject` method to see the rest of the definition.

Most often you'd use programmatic styling if you have to dynamically change values as a reaction to something in your application, such as animating

the size of a widget.

An issue with this approach is that if you need to change values, you have to recompile and distribute your code, and `UIObject` provides only a few methods for this. Wider access is given through the DOM, as you'll see next.

3.8.2. Low-level styling

If you need fine-grain control over styling in your code, for example, in DOM-based animation, then you can access style attributes using the `getStyle` method on a widget's element. You can set generic properties using the `setProperty` method or specific properties such as `setOpacity`. You use both methods to style the Search button in `BasicProject` (see the `styleButtonUsingDOM` method):

```
search.getElement().getStyle().setProperty("border", "2px solid");
search.getElement().getStyle().setOpacity(0.5);
```

One benefit of using the DOM class methods, such as `setOpacity`, is that you don't have to care how the browser does it (which incidentally is differently for IE6/7 compared with FireFox, Opera, IE8, and others). GWT takes care of that for you.

If you use the generic approach, remember that the property must be given in `camelCase`, that is, the CSS `background-color` must be written as `backgroundColor` (development mode will remind you about this with an assertion error). Note that this prevents you from using browser-specific CSS extensions because they include a hyphen in their names.

Figure 3.9 shows the before and after images of the button styling for `BasicProject`.

Figure 3.9. Styling a regular HTML button (left) in GWT and using Cascading Style Sheets (right)

You might see legacy code using the `DOM.setStyleAttribute` method to set style through the DOM. It's better to use the `Element` approach given here because it's more efficiently compiled.

Setting style in code is useful in certain circumstances, but like all good web design, it's often better to do it through Cascading Style Sheets to provide flexibility. This is easily supported by GWT.

3.8.3. Cascading Style Sheets

Most widgets have a style name associated with them when they're created. For example, an image has the style name `gwt-Image` associated with it. The Javadoc of GWT shows you what style names are associated with each widget by default.

Tip

How do you style a widget using CSS? Check out the Javadoc for the widget class, which will tell you the style names attached to various component parts of the widget/panel. For example, a `Button` has the standard style name `.gwt-Button`.

If you give a definition for the widget's default style name in a linked CSS file, then all widgets of that type will use that style. In `BasicProject` you put a simple border around the logo `Image` by adding the following to BasicProject.css:

```
.gwt-Image{
   width: 80%;
   height: 60px;
```

```
    margin: 25px;
    border: 1px black solid;
}
```

Often you don't want all widgets of the same type to have the same style. That's solved by using the `setStyleName` method on a widget to give an individual name to a widget. You use this approach for the feedback panel in `BasicProject`, by using

```
feedback.setStyleName("feedback");
```

and then refer to this name in the stylesheet (note the dot at the start of the definition):

```
.feedback{
    color: white;
    height: 150px;
    position: absolute;
    right: 0%;
    top: 35%;
}
```

You can use the methods `addStyleName` and `removeStyleName` to add styles to and remove them from a widget. For the feedback tab you want it to have a different style when the mouse is over it than when it's not. That's achieved by putting a `MouseOverHandler` on the panel to add a style called "active" and a `MouseOutHandler` that removes that active style and replaces it with a "normal" style. Both styles are defined in the stylesheet.

GWT has a mechanism for applying optimization techniques even to CSS—a `CssResource`. We'll look at that in chapter 5 and see how it supports runtime substitution, conditionals, minification, selector merging, and other things.

It would be a pain to have to create a stylesheet component for every widget you use. Luckily, GWT comes with three built-in themes you can select from.

3.8.4. GWT themes

In the module file you can tell GWT to use a particular theme for CSS styling. This saves you from having to define stylesheet entries for every widget you're using; you can see the results of the standard theme in figure 3.10.

Figure 3.10. The result of using a GWT theme on the `TabLayoutPanel`

You can choose from four themes out of the box. For the example application we've put them all in the module file, but we commented out two of them. If you want to change themes in the example, comment out the existing one and uncomment the one you wish to try:

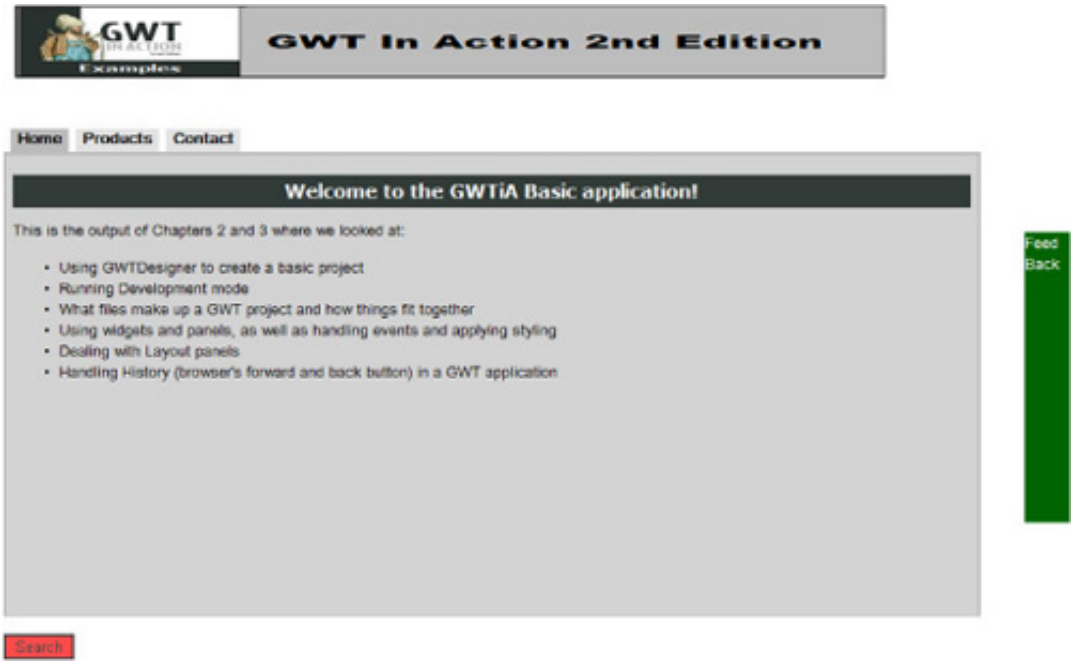
```
<inherits name='com.google.gwt.user.theme.standard.Standard'/>
<!-- inherits name='com.google.gwt.user.theme.chrome.Chrome'/ -->
<!-- inherits name='com.google.gwt.user.theme.dark.Dark'/ -->
<!-- inherits name='com.google.gwt.user.theme.clean.Clean'/ -->
```

You could generate your own themes—the simplest approach is to copy an existing one and adapt as needed.[6]

[6] An alternative is to use an online theme generator, such as the one found at http://gwt-theme-generator.appspot.com/.

When you apply all the approaches we've been talking about in this chapter, you get the `BasicProject` example to look like figure 3.11.

Figure 3.11. Final styled application showing all widgets, panels, pulled-in content, and styling in place—compare it with figure 3.8 to see how far we've come.



We touched on it a little earlier, but now let's briefly review how GWT helps you protect your application.