**Associating CSS Files**

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

* Using a <link> tag in the [host HTML page](http://www.gwtproject.org/doc/latest/DevGuideOrganizingProjects.html#DevGuideHostPage).
* Using the <stylesheet> element in the [module XML file](http://www.gwtproject.org/doc/latest/DevGuideOrganizingProjects.html#DevGuideModuleXml).
* Using a [CssResource](http://www.gwtproject.org/doc/latest/DevGuideClientBundle.html#CssResource) contained within a [ClientBundle](http://www.gwtproject.org/doc/latest/DevGuideClientBundle.html).
* Using an inline [<ui:style>](http://www.gwtproject.org/doc/latest/DevGuideUiBinder.html#Hello_Stylish_World) element in a UiBinder template.

Modern GWT applications typically use a combination of CssResource and UiBinder. Older applications should use only one of the first two choices.

**Client Bundle**

The resources in a deployed GWT application can be roughly categorized into resources to never cache (.nocache.js), to cache forever (.cache.html), and everything else (myapp.css). The [ClientBundle](http://www.gwtproject.org/javadoc/latest/index.html?com/google/gwt/resources/client/ClientBundle.html) interface moves entries from the everything-else category into the cache-forever category.

1. [DataResource](http://www.gwtproject.org/doc/latest/DevGuideClientBundle.html#DataResource)
2. [TextResource and ExternalTextResource](http://www.gwtproject.org/doc/latest/DevGuideClientBundle.html#TextResource)
3. [ImageResource](http://www.gwtproject.org/doc/latest/DevGuideClientBundle.html#ImageResource)
4. [GwtCreateResource](http://www.gwtproject.org/doc/latest/DevGuideClientBundle.html#GwtCreateResource)
5. [CssResource](http://www.gwtproject.org/doc/latest/DevGuideClientBundle.html#CssResource)
6. [CssResourceCookbook](http://www.gwtproject.org/doc/latest/DevGuideClientBundle.html#CssResourceCookbook)

**CSSResource**

**Overview**

1. Write a CSS file, with or without GWT-specific extensions
2. If GWT-specific extensions are used, define a custom subtype of CssResource
3. Declare a method that returns CssResource or a subtype in an ClientBundle
4. When the bundle type is generated with GWT.create() a Java expression that evaluates to the contents of the stylesheets will be created
   * Except in the simplest case where the Java expression is a string literal, it is generally not the case that a CSS file could be generated into the module output
5. At runtime, call CssResource.ensureInjected() to inject the contents of the stylesheet
   * This method is safe to call multiple times, as subsequent invocations will be a no-op
   * The recommended pattern is to call ensureInjected() in the static initializer of your various widget types

### Features

#### Constants

@def small 1px;

@def black #000;

border: small solid black;

* The parse rules make it difficult to use delimiting tokens for substitutions
* Redefining built-in sizes allows users to write plain CSS to draft a style and then tweak it.
* Suggest that users use upper-case names, similar to static final members.
* Any legal property value or expression may be used with @def
* @def rules that define a single numeric value may be accessed in a manner similar to obfuscated class names by defining an accessor method on the CssResource type that returns a primitive numeric value.

interface MyResources extends CssResource {

int small();

}

* + Calling small() would return the value 1.
* @def rules can be accessed as a String as well. You can retrieve the two definitions above with:

interface MyResources extends CssResource {

String small();

String black();

}

* + Calling small() returns "1px"
  + Calling black() returns "#000"
* The Generator will not allow you to declare an @def rule with the same name as a class, unless you annotate method to retrieve the class with the @ClassName annotation.

@def myIdent 10px;

.myIdent {

...

}

interface MyResources extends CssResource {

String myIdent();

@ClassName("myIdent")

String myIdentClass();

}

* + Calling myIdent() returns @def value "10px"
  + Calling myIdentClass() returns the obfuscated class name for .myIdent

### Optimizations

#### Basic minification

Basic minification of the CSS input results in the minimum number of bytes required to retain the original structure of the input. In general, this means that comments, unnecessary whitespace, and empty rules are removed.

.div {

/\* This is the default background color \*/

background: blue;

}

.empty {}

would be transformed into

.div{background:blue;}

#### Selector merging

Rules with identical selectors can be merged together.

.div {prop: value;}

.div {foo: bar;}

becomes

.div {prop:value;foo:bar;}

However, it is necessary that the original semantic ordering of the properties within the CSS is preserved. To ensure that all selector merges are correct, we impose the restriction that **no rule can be promoted over another if the two rules define a common property**. We consider border and border-top to be equivalent properties, however padding-left and padding-right are not equivalent.

Thus

.a {background: green;}

.b {border: thin solid blue;}

.a {border-top: thin solid red;}

cannot be merged because an element whose CSS class matches both .a and .b would be rendered differently based on the exactly order of the CSS rules.

When working with @if statements, it is preferable to work with the form that operates on deferred-binding properties because the CSS compiler can evaluate these rules statically, before the merge optimizations. Consider the following:

.a {

background: red;

}

@if user.agent safari {

.a {

\-webkit-border-radius: 5px;

}

} @else {

.a {

background: url('picture\_of\_border.png');

}

}

In the safari permutation, the rule becomes .a{background:red;\-webkit-border-radius:5px;} while in other permutations, the background property is merged.

#### Property merging

Rules with identical properties can be merged together.

.a {background: blue;}

.b {background: blue;}

can be transformed into

.a,.b{background:blue;}

Promotion of rules follows the previously-established rule of not promoting a rule over other rules with common properties.