

## Chapter 9: Incorporating Images and CSS

In this chapter we start a new Seaside application that will incorporate more of the complexities normally found in a sophisticated web application. The primary focus of this chapter is how to incorporate CSS (for layout and style) and images in your web application.

Imagine that your child has been recruited to play on the best youth football (*soccer* for you Americans) team in the region, *Los Boquitas*. (The first version of this tutorial was done in Buenos Aires, the home of *Club Atlético Boca Juniors*. *Boca* is Spanish for 'river mouth' and the club is in a neighborhood on the mouth of the Matanza River. The idea of naming a children's team 'Little Mouths' seemed appropriate.) All the parents are expected to be involved and instead of coaching on the field, you have agreed to create a web site to manage some team information. After a review of various technologies, you have decided to use Seaside.

Following is a screen shot of the basic structure, showing various pieces of a web site (header, sidebar, main, image, footer), that we will build.



This layout is, of course, completely arbitrary. Web page layout is best done by a designer and that designer should provide sample HTML and the CSS to go with the HTML. As a programmer, your job is to translate the HTML into Smalltalk code and reference the supplied CSS.

1. First we will define a home page for the site. We will use 'LB' (for *Los Boquitas*) as the prefix for our classes. While some Smalltalk dialects support namespaces (Cincom and GemStone), this is a convention that helps avoid name conflict when porting code across dialects. Launch Squeak and enter the following class definition:

```
WAComponent subclass: #LBMain
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'LosBoquitas'
```

- a. To provide a description that will show on the dispatcher page, add this *class*-side method to LBMain (remember to click the 'class' button below the class list to work with class-side methods):

```
description
```

```
^'Keep track of children''s football (soccer) team'.
```

- b. Previously we have registered our application by evaluating an expression in a workspace. As the expression gets more complex, it becomes desirable to put the expression in a method and then just call it from a workspace. This allows us to manage the code with our source code management tools (tracking versions, etc.). An added benefit is that when a class is first loaded using the code management tools, the message 'initialize' is sent to the class. This means that our application can be automatically registered when it is loaded into a new Smalltalk object space.

In this method we register the name (boquitas) and also specify that we want to use a WASession as the session class. This is the default and it expires a session that is inactive for a configurable period (the default is 10 minutes). If you get a warning dialog when you save the code then click 'Override' to confirm.

```
initialize
"
  LBMain initialize.
"
super initialize.
(self registerAsApplication: 'boquitas')
  preferenceAt: #sessionClass put: WASession;
  yourself.
```

- c. Now we are ready to add some content. Add this *instance*-side method to LBMain (click on the 'instance' button to switch from the class side back to the instance side):

```
renderContentOn: canvas

  canvas heading
    level: 1;
    with: 'Los Boquitas Soccer Team'.
```

2. Now initialize your component by executing the following in a workspace:

```
LBMain initialize.
```

3. Finally, start at the dispatcher page, <http://localhost:8080/seaside>, and follow the link to boquitas. If your application is not visible, then go back and see if you put the proper methods on the class side of LBMain. If they are on the instance side, then things will not work.
4. Now we will start the process of adding an image.
  - a. Modify the render method in LBMain so that there is an image tag but it points to a file that does not exist:

```
renderContentOn: canvas

  canvas heading
    level: 1;
    with: 'Los Boquitas Soccer Team'.
  canvas image
    url: 'boquitas.jpg';
    yourself.
```

- b. Try viewing the page and notice that the image does not display. Depending on your browser, a placeholder might be displayed. At a minimum we need some alternate text to be displayed when the image is missing. Modify the render method again:

```
renderContentOn: canvas

  canvas heading
    level: 1;
    with: 'Los Boquitas Soccer Team'.
  canvas image
    altText: 'children playing soccer';
    url: '/images/boquitas.jpg';
    yourself.
```

- c. Try viewing the page and see if the alternate text is displayed. In Safari the text is not displayed, but this is a feature, not a bug! Technically, the official definition of the alt attribute is that it is for user agents (browsers) that cannot display images (such as

screen readers for visually impaired users). Since Safari is capable of displaying images, the alt attribute is ignored.

- d. Now we could update the link to reference a site that does have the picture. (If you are not connected to the Internet, skip to step #5.) Modify the render method to point to an external server with the file. (If you are running a local server, you can download the file from [seaside.gemstone.com](http://seaside.gemstone.com) and install it in your local images directory.)

```
renderContentOn: canvas

  canvas heading
    level: 1;
    with: 'Los Boquitas Soccer Team'.
  canvas image
    altText: 'children playing soccer';
    url: 'http://seaside.gemstone.com/images/boquitas.jpg';
    yourself.
```

- e. View the page and verify that the image shows.
5. While having an external server provide static data (such as images) is efficient (reducing the load on your local server—in our case the Squeak VM), it does mean that your application is not completely self-contained. Sometimes for development, testing, or demos it is nice to have even static data served from Smalltalk (and as long as you can't measure the performance impact, why not?). To support serving static data, Seaside provides an abstract class, `WFileLibrary`.
    - a. Save a copy of the image to a local directory. You can do this from your web browser by right-clicking on the image and selecting 'Save image as...' (the exact menu command will differ based on your web browser). Alternatively, copy the file from a CD/DVD if included.
    - b. Create a subclass of `WFileLibrary` to handle static data for our application:

```
WFileLibrary subclass: #LBFileLibrary
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: ''
  category: 'LosBoquitas'
```

- c. Add the downloaded image to the file library by evaluating an expression in a workspace that includes the path to the file (this will, of course, differ based on your machine).

```
LBFileLibrary addFileAt: 'C:\temp\boquitas.jpg'
```

- d. Now using the System Browser, note that a new method ('boquitasJpg') has been added to the instance side of LBFileLibrary. Modify the render method to reference the new library. Notice how the class LBFileLibrary understands the '/' message and returns a reference to the static file.

```
renderContentOn: canvas

  canvas heading
    level: 1;
    with: 'Los Boquitas Soccer Team'.
  canvas image
    altText: 'children playing soccer';
    url: LBFileLibrary / 'boquitas.jpg';
    yourself.
```

- e. If you view source in your web browser, you can see that Seaside generated the following element for the page:

```

```

6. Adding a page title.
  - a. Note that the page is simply titled "Seaside" rather than something more descriptive. Add the following instance-side method to LBMain:

```
updateRoot: anHtmlRoot

  super updateRoot: anHtmlRoot.
  anHtmlRoot title: 'Los Boquitas'.
```

- b. View the page again and note that it now has a title.

7. Creating multiple areas.

- a. The typical web site has a header, a side-bar, a main content area, and a footer. We will now add these pieces by modifying the render method again. Note that the #'with:' message is the last one sent to each element since this causes the HTML tag to be written and closed.

```
renderContentOn: canvas

  canvas div
    id: 'allcontent';
    with: [
      canvas div
        id: 'header';
        class: 'section';
        with: [
          canvas heading
            level1;
            with: 'Los Boquitas Soccer Team'.
        ].
      canvas div
        id: 'main';
        class: 'section';
        with: [
          canvas image
            altText: 'children playing soccer';
            url: LBFileLibrary / 'boquitas.jpg';
            yourself.
        ].
      canvas div
        id: 'sidebar';
        class: 'section';
        with: [
          canvas heading
            level2;
            with: 'Sidebar'.
        ].
      canvas div
        id: 'footer';
        class: 'section';
        with: [
          canvas text: 'Copyright (c) ' , Date today year printString.
        ].
    ].
```

- b. View the page and confirm that the various pieces exist. Note, however, that they do not have any formatting.

8. Until recently, the typical way of doing page layout was to use a table. This approach is no longer recommended and the better approach is to use Cascading Style Sheets (CSS). In chapter 5 we saw that a 'style' method on our component could provide style. This is useful for simple experiments, but Seaside provides alternatives.
  - a. Add the following method to the instance side of LBFileLibrary:

```
boquitasCss
^'body {
  font-family:      Georgia, "Times New Roman", Times, serif;
  font-size:        small;
}

#allcontent {
  width:            770px;
  padding-top:      5px;
  padding-bottom:   5px;
  background-color: #fff0d0;
  margin-left:      auto;
  margin-right:     auto;
}

*.section {
  background-color: #ffeabf;
}

#header {
  margin:           10px;
}

#main {
  padding:          15px;
  margin:           0px 10px 10px 10px;
  width:            550px;
  float:            right;
}

#sidebar {
  padding:          15px;
  margin:           0px 600px 10px 10px;
}

#footer {
  color:            #204670;
  text-align:       center;
  padding:          15px;
  margin:           10px;
  clear:            right;
}'
```

- b. Modify the root component to reference the style sheet:

```
updateRoot: anHtmlRoot

super updateRoot: anHtmlRoot.
anHtmlRoot title: 'Los Boquitas'.
anHtmlRoot link
  type: 'text/css';
  beStylesheet;
  addAll;
  url: LBFileLibrary / 'boquitas.css';
  yourself.
```

9. Let's refactor the render code so that it is more modular.
  - a. Add the following four methods:

```
renderHeaderOn: canvas

canvas div
  id: 'header';
  class: 'section';
  with: [
    canvas heading
      level1;
      with: 'Los Boquitas Soccer Team';
      yourself.
  ];
  yourself.
```

```
renderMainOn: canvas

canvas div
  id: 'main';
  class: 'section';
  with: [
    canvas image
      altText: 'children playing soccer';
      url: LBFileLibrary / 'boquitas.jpg';
      yourself.
  ];
  yourself.
```

```
renderSidebarOn: canvas

canvas div
  id: 'sidebar';
  class: 'section';
  with: [
    canvas heading
      level2;
      with: 'Sidebar';
      yourself.
  ];
yourself.
```

```
renderFooterOn: canvas

canvas div
  id: 'footer';
  class: 'section';
  with: [
    canvas text: 'Copyright (c) ' , Date today year printString.
  ];
yourself.
```

b. And now modify the renderContentOn: method to call these new methods:

```
renderContentOn: canvas

canvas div
  id: 'allcontent';
  with: [
    self
      renderHeaderOn: canvas;
      renderMainOn: canvas;
      renderSidebarOn: canvas;
      renderFooterOn: canvas;
      yourself.
  ].
```

c. View the application in a web browser to confirm that it displays the page shown at the beginning of this chapter.

10. Save your Squeak image.