Adding *Style* to BBx® Web Apps With Custom CSS

Cascading Style Sheets, referred to as CSS, have been used since the mid-90's and over time have become the de facto way to define how a website should look. The primary goal of CSS has always been to separate the presentation (how the page looks) from the content (the actual text or article in the web page). There are many advantages to this as changes to one CSS file can propagate to an entire website, resulting in reduced development and testing time, greater consistency across the website, reduced ongoing maintenance, and so on.

CSS offers many benefits beyond that primary goal as CSS has continued to evolve, and add greater functionality, due in part to users and website designers demanding new features and capabilities so that web pages can compete with traditional desktop applications. Although CSS has conventionally been used to style web pages, BBx® developers can now reap the benefits of CSS when deploying their applications in BASIS' Browser User Interface (BUI).

By Nick Decker
Engineering Supervisor

So Why is CSS Exciting?

When a BBx application runs in BUI, it is executing on the end user's browser and making use of the standard HTML, JavaScript, and CSS capabilities that come with the platform. BASIS includes a default style sheet for use with BUI apps, but has designed BUI configuration to be open to overriding this CSS file with a custom sheet. Ultimately, this means that anyone now has the power to easily change a BBx application's look and feel using a popular, open, independent, and freely usable standard.

Before digging into the details, let us take a quick look at a BBj® BUI app that utilizes various aspects of CSS to create a rich UI. Because a picture is worth a thousand words, comparing the app visually with and without the custom CSS file says volumes about its capabilities and is the best illustration of how much CSS can affect the presentation. **Figure 1** compares the same BUI app running on an iPhone – one with custom CSS and one without.

The images in **Figure 1** on the next page, illustrate many of the formatting options that are available via CSS and drive home the point that we can manipulate the appearance of the content without modifying the content itself. In other words, it is the same BBx program running in both scenarios; both using a BBjStaticText control to display the "BUI Tip Calculator" title at the top of the screen. The CSS employed in the second example uses the entry in the stylesheet definition file and shown in **Figure 2** to format the static text and turn it into a fancier title. > >

```
/* Customized top bar for title */
.topGradientBar {
    background: -webkit-gradient(linear, left top, left bottom,
        color-stop(0.0, #8b98ba), color-stop(0.45, #4764a0),
        color-stop(0.46, #3c5899), color-stop(1.0, #314a82)) !important;
    font-family: Tahoma, Helvetica !important;
    font-size: 30px !important;
    font-weight: bold !important;
    line-height: 46px !important;
    text-align: center !important;
    color: #eceff5 !important;
    text-shadow: rgba(255,255,255, 0.75) 0px 1px 1px,
                 rgba(0,0,0,.9) 0px -1px 1px !important;
    -webkit-box-shadow: 0px 0px 4px rgba(0, 0, 0, .95) !important;
}
```

Figure 2. The CSS definition for the BBjStaticText control used for the title

Figure 1. A BUI app running without custom CSS (left) and with custom CSS (right)

Dissecting the CSS Example

The CSS entry starts off with a comment on line 1 that documents the function of that section of the file. Line 2 begins the actual definition for the CSS Selector, which is what CSS uses to define elements. The selector name is important as that is what is used to apply the new style to one or more BBj Controls. Selector names are unique and user-defined, but be aware that BASIS has reserved selectors and special selector names that correlate to control names. For example, if we named this selector **`.BBjButton`**, then the style would automatically apply to all BBjButton controls in the app. For more information on reserved selectors and BUI CSS in general, see the CSS API documentation. Because this is not a reserved selector, we can optionally apply it to any control with the **`addStyle()`** method.

Lines 3-14 in the example list a number of properties and their associated values that comprise the definition of the custom selector. CSS properties are generally well named, and **`font-size`** on line 7 serves as a good example that shows it should not be difficult to figure out what effect the property will have on the control. The BASIS IDE's built-in syntax coloring also comes in handy, making it easy to discern which properties are standard and which are proprietary. The properties in **Figure 2** are responsible for adding a background gradient to the control; setting the font, size, weight; vertical positioning, horizontal alignment, and color of the text. They even create an embossed effect for the title by applying a semi-transparent white shadow below the text and a semi-transparent black shadow above the text. This is just a taste of what CSS offers, as developers now have command over a multitude of presentation properties and can easily transform the look of their applications without making any changes to the program's source code.

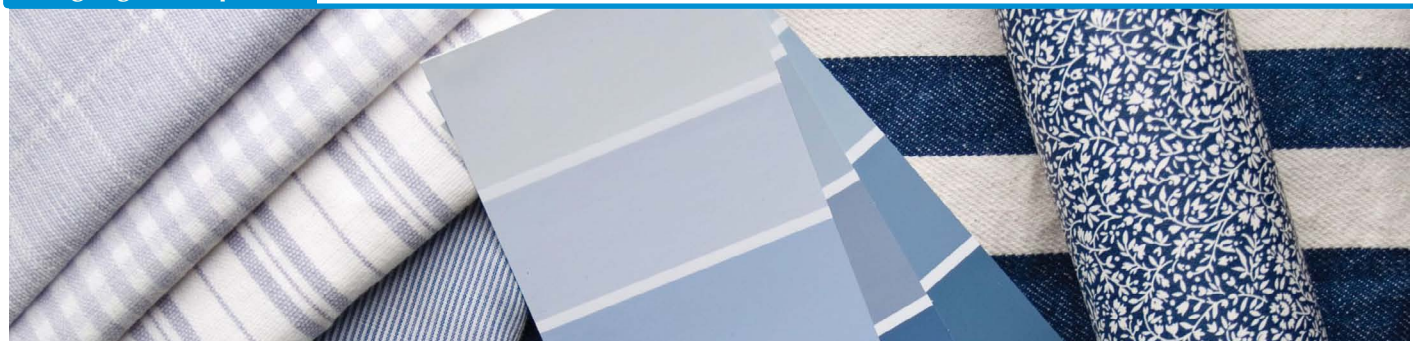
One Size Doesn't Have to Fit All

One of the selling points for CSS is that developers can easily change the look and feel of their BUI program without modifying the BBj source code. The previous example changed the look dramatically, but what about a special requirement or need such as an employee that is color blind or vision impaired? Or perhaps a company wants to brand their application with their corporate colors and logo? CSS offers solutions to all of these scenarios by allowing developers to create as many different themes for their applications as necessary. Enterprise Manager makes it easy to register a BBj program as a BUI app, and it is just as easy to register the same application multiple times with unique style sheets.

With CSS, simply deploy the same BUI app to different clients, but with each client having their own custom URL and look and feel for the app. Now adding a new color scheme for a color blind user is possible with very little effort. Additionally, one can improve legibility for those with eyesight limitations by increasing the font size, specifying higher-contrast color schemes, and clearly highlighting selected and active controls – all with just a few lines of CSS. Adding hover effects also enriches the user experience and gives immediate feedback for mouse movements, improving usability and adding an extra level of interactivity to your app.

Improving Upon the Defaults

Enhancing user interactivity is one area in which CSS excels as developers have so much influence on the appearance of the controls on a form. In addition to the CSS selectors mentioned previously, CSS has the concept of Pseudo-Classes/Elements that modify or add special effects to existing selectors. These pseudo-elements make it possible to further customize or differentiate controls in various states, such as the text control on a form that currently has keyboard focus, or the button that the user is clicking. Calling attention to these controls by changing their style enhances the user's sense of direct manipulation and is a great way to improve the user experience. >>



A common use case for customizing the look of a particular control's default state would be to enhance legibility for read-only or disabled text controls. Different operating systems have their own individual way of coloring these controls, but this sometimes results in making the application harder to use. Sometimes the differentiation between an editable and read-only control is so slight that users may not notice the difference and become frustrated when the application thwarts their attempts to edit the control. Other times the disabled version of a text control results in dark gray text on a medium gray background; sure, it is easy to tell that the application has disabled the control but the color scheme makes the text virtually impossible to read.

Using CSS, we can easily remedy these scenarios and even provide multiple solutions in the same application via CSS styles given the users' preference. We can do this on an application-wide level using two aforementioned BASIS reserved selectors, **`.bbj-disabled`** and **`.bbj-readonly`**. In addition, we could also specify multiple classes in the same selector to further filter which controls the selector applies to. For example, the selector

```
.bbj-disabled.myCustomButton { … }
```

will only affect controls that have previously had the "myCustomButton" style applied and are currently disabled. Multiple class selectors make it easy to isolate a specific set of controls and result in the selector overriding less specific definitions.

Other CSS Benefits

The BBjStaticText control in the BUI demo now looks a lot better, and one of the benefits of stylizing with CSS is that it scales extremely well. Unlike graphic images that lose quality when the user zooms in their browser view, CSS looks even better when zoomed. To illustrate this difference, **Figure 3** shows a side-by-side comparison of a zoomed-in BBjButton defined both as a graphic image, and by using CSS without images. Notice how the text and the button itself scale dramatically better on the right with CSS compared to using an image of the button, making the button sharper and easier to read.

Figure 3. A button defined with an image (left) compared with CSS (right)

This extra quality does not come with much of a cost, either. Although there are ways to optimize the use of graphic images in CSS, such as using sprites, image-free CSS is very lightweight. The client requires only the selector definition and then renders the object accordingly. This can reduce network overhead as compressed CSS text files have a small footprint and require much less bandwidth compared to what may amount to numerous large image files. With CSS, the client web browser is now responsible for the layout and rendering of many graphics, but that too may be optimized. Certain platforms, such as iPhone's iOS, offload some of the more tedious CSS rendering tasks to the dedicated GPU (graphics processing unit). This results in smooth animations such as fades, zooms, translations, scaling, etc., while reducing the load on the machine's CPU. The final result is higher quality renderings and reduced network traffic – a double whammy – making CSS a great way to increase the visual appeal of your apps while increasing performance at the same time.

Taking CSS Further - Custom Controls

With a little ingenuity and some fancy CSS, we can create a "custom control" from one or more of the traditional BBj controls. There are several reasons for doing this, not the least of which is "because we can!" Creating custom controls can add to the appeal of your web app and may even enhance usability and blend in better with the purpose of your app. As an example, the "BUI Tip Calculator" adds and removes CSS styles to regular BBjButtons to create both a star rating system and a graphical way to specify the number of guests at the restaurant. **Figure 4** shows a section of the app where the user has set the number of dinner guests to three.

Figure 4. Specifying the number of guests in the dinner party

Technically, we are just looking for a way to get a numerical value from the user and there are several controls that could accomplish that task: a BBjEditBox, BBjInputN, BBjSlider, to name a few. In this case though, a custom control makes a lot of sense, and is a better choice compared to the traditional controls. For starters, we designed the app to run on a smartphone, like the iPhone, a touch-enabled device with a small screen. Our custom control is perfect for this environment since the user can specify the number of guests via a single tap. In comparison, if we had > >

used an edit control, then the user would first tap on the control causing the phone's keyboard to slide up from the bottom of the screen. The user would then have to find the desired numerical key on the keyboard and tap it. A third and final tap would be required to dismiss the keyboard and return to the app. This process is inefficient, cumbersome, and requires more dexterity from the user which distracts from the 'flow' of the app and user experience in general. Lastly, a text control can be a bit boring, especially in the iPhone world where app developers focus on designing appealing applications with alluring user interfaces. Apple even provides a 150+ page document covering best practices for Mobile Human Interface Guidelines, which contains sections with titles like "Delight People with Stunning Graphics" to ensure a high level of quality.

Another example of a custom control is the loading indicator that displays when the "BUI Weather" portion of the same demo program refreshes its data. **Figure 5** shows how this custom control notifies the user that the app is busy loading new weather data.

Figure 5. Custom Busy Indicator

BBjStaticText controls, along with some CSS to define the appearance and provide animations, work together to create this custom control. This is another case where a traditional control, such as a BBjProgressBar, would be a reasonable candidate. However, since we want our app to look and feel more like a native iPhone app, we can mimic some of the more popular iOS elements and behaviors such as an alert message box or a HUD-style busy indicator. Our custom control does just that and relies on CSS to provide fade-in and fade-out animations, a translucent backdrop for the control with rounded corners, and an animated rotating busy indicator. The result is the same in that the user is still notified that the app is busy and will not accept any user input, but thanks to CSS, it is a pleasant experience that the user is accustomed to because the BUI app looks and acts like a native app.

Summary

We have now introduced you to the many facets of CSS and how it can single-handedly improve your app's look, functionality, and even performance. You gain complete control over each screen element's appearance including spacing, alignment, positioning, font, size, weight, color, opacity, shadows, and more. CSS also reduces your development time when you begin to share CSS files between apps, enforcing consistency and reducing maintenance.

With CSS, you can also brand applications on a per-customer basis, provide an updated look-and-feel without touching the BBj source code, and even provide multiple themes and color schemes within the same app to improve the user experience. CSS makes it possible to create custom control interfaces that mimic native controls on other platforms, increasing the ease-of-use and familiarity for users. Lastly, CSS is a free open standard that is pretty easy to learn – especially with the hundreds of books, web pages, online tutorials, and examples available on the topic.

So what are you waiting for? Add some *Style* to your BBx web apps today! ■

- For more information, review the online documentation for information on
  - BBjStaticText
  - BBjButton
  - CSS API
- Read about Pseudo Classes/Elements
- Check out Apple's "Mobile Human Interface Guidelines"