Forerunner CSS Framework Guidelines

Author: Jon Totton, July 2, 2013

# Summary

This guideline is being written to form the basis to standardize the definition and use of CSS classes. It will cover where to define, how to name and under what circumstance classes should be used.

# Classes and Styles

* In general, all UI in the forerunner SQL Server Report package will use classes to “style” the look and feel of the UI.
* All classes will be named according to the [Naming Conventions](#_Naming_Conventions) section below.
* All class definitions will be added to the files according to the [Where to Put Style Definitions](#_Where_to_Put) section below.
* Classes will be defined in Cascading Style Sheet (I.e., .css) files and not directly in .js files.
* “Style” attributes that are defined directly on html elements should NOT be used. If you do choose to use a style attribute a comment should be added to explain why it was used.
  + A prominent exception to this guideline is the Report Renderer widget. The use of styles in the Report Renderer case is a reasonable choice for what it does. It is interpreting RPL data and converting that data into styles that have a hi-fidelity match with the intent of the RPL and not a stylized interpretation.
* Classes should be defined with appropriate specificity
  + As an example, a widget should specify all tag selector classes (e.g., div) as children of the top level widget class; so for the toolbar widget:

.fr-toolbar {

}

.fr-toolbar div {

}

The classes above would style only the <div> elements defined under the toolbar widget.

# Where to Put Style Definitions

* In general all public facing objects defined in the forerunnersw.com SDK will be defined as jQuery widgets. For each widget there should be one .js file. For each .js file containing a widget there should be one associated .css file of the same name
* The js and css files should have the same name as the widget.
  + Filename should be named with upper and lower case characters. E.g., the “reportViewer” widget should be defined in a file pair named js/ReportViewer.js, css/ReportViewer.css
* Javascript files should live under a “js” folder and Cascading Style Sheet files should live under a “css” folder.
* In the case where common base functionality in javascript, spans more than one widget, a base widget and associated .css file should be created to encapsulate the shared functionality.
  + An example is the toolbase widget. Currently toolbase is shared between the reportexplorertoolbar, toolbar and toolpane widgets.
* In the case where classes are core to the SDK and not inherently associated via a base class, a new file will be created called *forerunner-core.css*. Any class that is shared between two or more widgets not via a base widget should be defined in this file.

# Naming Conventions

All classes should be named using the following convention:

fr-<component>-<detail>-<postfix>

Where:

fr-

Literal, all forerunner classes will start with “fr-“

<component>

Is either the widget name (it is ok to abbreviate) or a core area of class definitions such as tool, container, state, etc.

<detail>

Is the detail indicating what the class is for.

<postfix>

Postfix is optional, when a postfix is needed, it will indicate additional detail such as direction, whether a class is an id, a class styles a button, etc. Common postfixes are as follows:

button

Used to indicate the style is for a button type of toolbar item.

id

Used to indicate that a class is used as a selector and does not have a class definition

tl, tr, bl, br, top, bottom, right. left, all

Used to indicate which part of the border a class applies to. Such as Top left, right, bottom left, right, etc.

Examples:

.fr-toolbar

Top level widget class.

.fr-toolbar div {

}

Stylize all <div> elements under the toolbar widget.

# SDK files

For the forerunner SDK all js files should be concatenated and minimized. For the associated .css files the strategy is different. That is because the intent is to allow the end users the ability to stylize the widgets via the individual .css files. We will however package up the .css file into the following structure that will make including the .css files easier:

*Forerunner-all.css*

This file will use the @import directive to include all the .css files used by the SDK. This will include *forerunner-core.css* as well as all individual *widget.css* files. This should be the only .css file and end user need link to.

*<widget>*.css

All widgets should define their own .css file and it should be imported into Forerunner-all.css.

To reduce the download footprint, the end user could choose to download only *forerunner-core.css* and the individual widget.css files that they actually use.

# Themes

Themes basically are simply linking to different versions (I.e., not the default set of .css files) of one or more classes when loading the page. The users can accomplish this by modifying the supplied .css files or creating new ones and then linking to the new style sheets on their pages.

In order for themes to work well there should be a strong set of core classes defined that are shared among the widgets

# forerunner-core.css

A set of core classes should be broken out of the individual .css files and common definitions should be shared via this file. This will enable more consistent styling among the forerunner widgets. As yet this common set of classes has not been defined.

# Future

When we have a solid core set of classes defined and shared by the widgets, we should consider building a theme roller like the one defined foe jquery-ui [here](http://jqueryui.com/themeroller/).

# Current problems

* The core classes have not been defined
  + This would make themes difficult to implement for our users because each widget would have to be modified individually.
* No consistency in the class names
* Forerunner-all.css and forerunner-core.css have not been defined yet
* There are some missing .css files (e.g., toolbase.css)
* It is probably worth a chat as to why we are not using the jQuery UI CSS Framework. We would certainly benefit from a solid foundational set of classes, the theme roller and when we use the jQuery UI widgets we would be styled in a consistent way. Worth a chat I think.