Paper 335-2010

SAS® Gets Flexy: Themes You Can't Resist

Amy Dull and Sherry Parisi, SAS Institute, Inc., Cary, NC

ABSTRACT

Have you heard the news? SAS® made a strategic decision to create its next generation of Web applications using the Adobe Flex platform. Flex is an open source platform designed for building rich Internet applications. There is also a Flex theme framework for coordinating colors, images, and text styling. Flex themes are integral to enhancing the overall SAS user experience. This paper explores how we developed a SAS corporate theme to create a stronger visual identity for the SAS product line. Two additional base themes allow customers to customize the SAS product line to meet branding needs, ensuring a unified look across the SAS suite.

INTRODUCTION

This paper will briefly discuss why Flex was chosen as the technology for all future Web-based product development at SAS. Along with this decision came the opportunity to theme our products consistently from the start! A theme in a general sense defines the overall look and feel of an application. It is designed using colors and graphics that are applied to common user interface components and layout containers. Standardization of icons, fonts, and to a lesser extent, effects, can also bring value when creating a theme, as it provides even stronger visual identity across a product or suite of products.

As SAS introduces more and more solution-based products to the market, we recognize that our customers will often be using more than one SAS application in their daily work. Having a consistent look and feel becomes very important so that the transition from one solution to the next is as seamless as possible. Unlike any time before in SAS history, executive R&D management called out theme support as a critical requirement for all products being developed. It is no longer optional; it is required.

So starting new with Flex, a dedicated team of developers and usability analysts have been working diligently to build out the infrastructure and design for theme support across all SAS products. Past customer feedback was a driving factor in the development process. We learned firsthand just how difficult it was (pre-9.2) for our Web-based customers to customize the look of a SAS product to better fit in with their corporate branding. This was important to them. Moving forward, we wanted to make sure we did not repeat our mistakes.

This paper will discuss in detail the process that the Flex Themes project team used to develop a new SAS Corporate theme for our Flex clients. Specifically, we will cover:

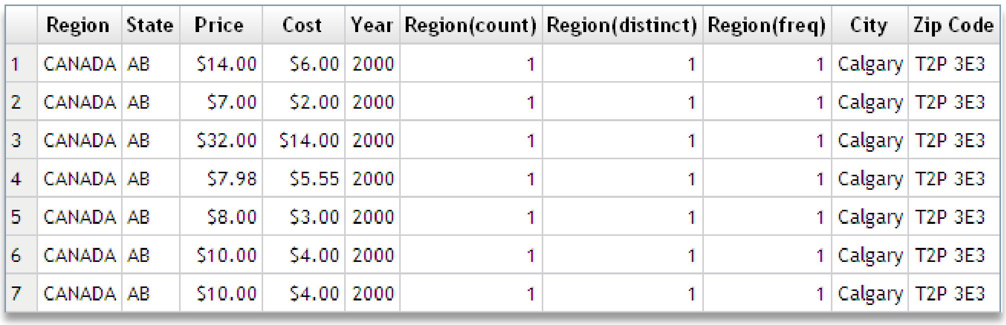
- Why SAS chose Flex technology.
- Tools we used (Adobe Photoshop, Adobe Illustrator, Adobe Flex Builder, etc.).
- Implementation of the SAS Corporate theme.
- Challenges we encountered.
- What's next for themes.

WHY FLEX?

User interfaces for SAS products and solutions have evolved over the years as technology has improved. From simple monochrome text to clean, Web-based designs, user interface design and usability have come a long way. The next generation of SAS products takes advantage of one of the latest and most dynamic user interface technologies: Adobe Flex.

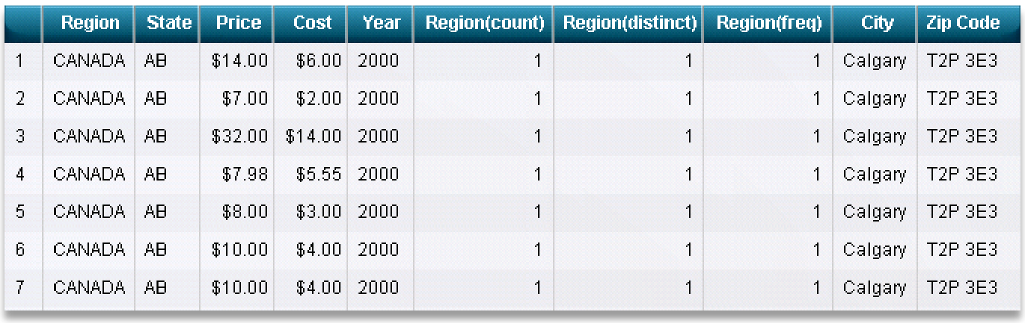
Flex offers an open-source framework for creating rich, interactive applications. A significant benefit of an application written in Flex is that the application runs within the Adobe Flash Player. While some view this as a drawback due to the requirement of installing Flash Player in addition to a Web browser, the fact that Flash Player nearly eliminates headaches due to browser incompatibilities is a major incentive to application developers. With a moderate learning curve, our Java developers were able to quickly begin working with Flex and ActionScript to create functional applications that looked good out of the box.

The previous implementation for SAS Web-based user interfaces relied on a combination of HTML, JavaScript, CSS, and images to provide functionality, look, and feel. Not only are these tools more limited in what you can achieve from a visual appearance perspective, but the development, testing, and maintenance burden in having to deal with various browser idiosyncrasies is costly. Changing the foundation for SAS user interfaces to Flex allows for less time spent fixing browser problems and more time spent building in increased interactivity, responsiveness, and a greater degree of visual impact. The following screen captures provide a comparison between user interface components from the “old” world of HTML and CSS and the “new” world of Flex. Figure 1 is the Web-based, relational table component, and Figure 2 shows the analogous Flex component—the DataGrid. Notice how the Flex table heading has much more depth and flair.



	Region	State	Price	Cost	Year	Region(count)	Region(distinct)	Region(freq)	City	Zip Code
1	CANADA	AB	$14.00	$6.00	2000	1	1	1	Calgary	T2P 3E3
2	CANADA	AB	$7.00	$2.00	2000	1	1	1	Calgary	T2P 3E3
3	CANADA	AB	$32.00	$14.00	2000	1	1	1	Calgary	T2P 3E3
4	CANADA	AB	$7.98	$5.55	2000	1	1	1	Calgary	T2P 3E3
5	CANADA	AB	$8.00	$3.00	2000	1	1	1	Calgary	T2P 3E3
6	CANADA	AB	$10.00	$4.00	2000	1	1	1	Calgary	T2P 3E3
7	CANADA	AB	$10.00	$4.00	2000	1	1	1	Calgary	T2P 3E3

Figure 1. Relational Table Rendered with HTML and CSS in the Default Web Theme



	Region	State	Price	Cost	Year	Region(count)	Region(distinct)	Region(freq)	City	Zip Code
1	CANADA	AB	$14.00	$6.00	2000	1	1	1	Calgary	T2P 3E3
2	CANADA	AB	$7.00	$2.00	2000	1	1	1	Calgary	T2P 3E3
3	CANADA	AB	$32.00	$14.00	2000	1	1	1	Calgary	T2P 3E3
4	CANADA	AB	$7.98	$5.55	2000	1	1	1	Calgary	T2P 3E3
5	CANADA	AB	$8.00	$3.00	2000	1	1	1	Calgary	T2P 3E3
6	CANADA	AB	$10.00	$4.00	2000	1	1	1	Calgary	T2P 3E3
7	CANADA	AB	$10.00	$4.00	2000	1	1	1	Calgary	T2P 3E3

Figure 2. Flex DataGrid in the Corporate Flex Theme

In Figure 3, two states of the Web-based TabMenu are shown. The lines that define the borders of the tabs themselves are rendered using repeating images. Figure 4 shows the Flex TabBar, the analog to the TabMenu shown above. Again, notice that the Flex tabs have more depth, and the mouse hover appearance has much more visual impact, behaving much more like a rich-client application.

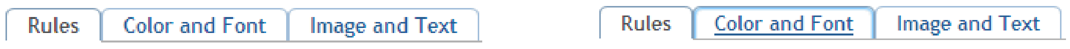


Figure 3. TabMenu Rendered with HTML and CSS in the Default Web Theme. The *Color and Font* tab on the right shows mouse hover appearance.

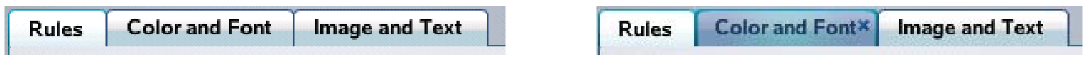
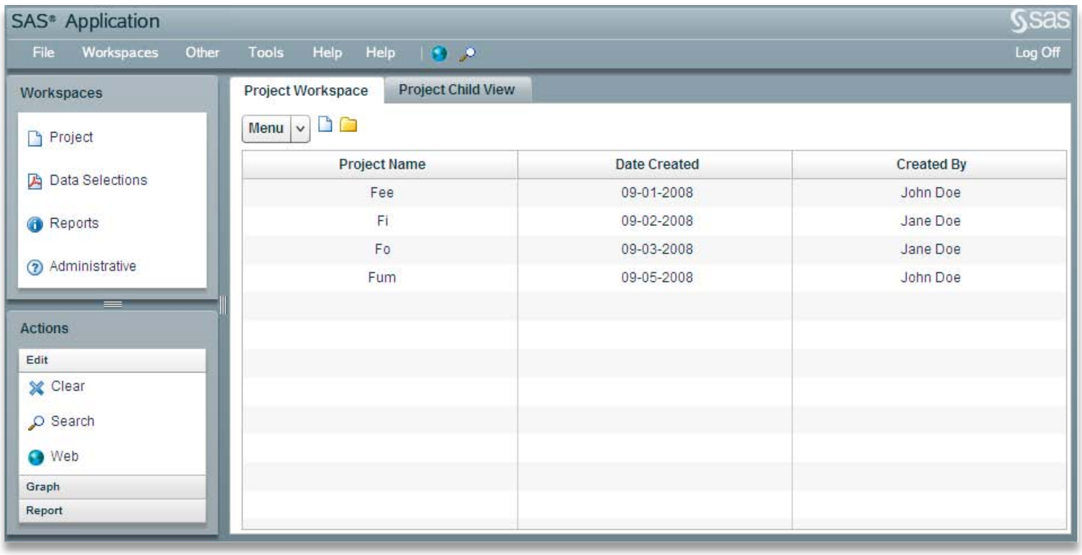


Figure 4. Flex TabBar in the Corporate Flex Theme. The *Color and Font* tab on the right shows mouse hover appearance.

Although Flex has been available since 2004, it has recently gained more of a following in the software development industry with the release of Flex 3 in early 2008. It becomes quite apparent which applications are implemented in Flex, because a majority of them have common elements and similar styling. This is due to their use of the default

Flex theme, “Halo,” that is applied to all Flex applications, and also to the absence of any customization of this basic theme. A basic Flex application using the Halo theme is shown below in Figure 5.



Project Name	Date Created	Created By
Fee	09-01-2008	John Doe
Fi	09-02-2008	Jane Doe
Fo	09-03-2008	Jane Doe
Fum	09-05-2008	John Doe

Figure 5. A Basic Flex Application with the Halo Theme

The Halo stylesheet (defaults.css) is provided with the Flex SDK source code. The stylesheet contains the base set of styles and default values for the Halo theme. It was helpful to use this stylesheet as a reference when considering the entire style hierarchy during the theme design process. Our goal in creating a new SAS Corporate theme was to use the pieces of the Halo designs that worked well, but to expand upon it to create a unique look and feel that belongs to the next generation of SAS products and solutions.

FLEX THEMES TOOLKIT

Before we begin discussing the details about the creation of our Corporate theme, let’s take a look at the new tools and resources that we investigated and that were of most use to us.

ADOBE FLEX BUILDER

Flex Builder (renamed "Flash Builder" for the Adobe Flex 4 release) is a development environment that enables application designers and developers to collaborate during the early stages of development. It is based on the Eclipse platform, so it has a familiar look and feel for many developers. It is available as a stand-alone application or as an Eclipse plug-in. A key feature of Flex Builder is the ability to view and edit application code and styles in either Source or Design mode. Source mode displays the source code for a particular class or CSS file in a color-coded text editor. This is most useful to software developers and programmers. Design mode provides a friendlier graphical interface that varies based on the content of the file. If the file contains MXML, Design mode represents the code as a wireframe version of the components within the overall layout. Users can drag and drop components and containers, adjusting layout and proportion with the mouse. If the file contains CSS, a sample component is displayed that shows its various states with the defined styles applied. Properties of styles or components in Design mode can be modified via drop-down menus, color choosers, and input fields. Design mode is especially useful to designers who want to put together a quick prototype or quickly view the effects of a particular style change.

ADOBE FLASH, ADOBE ILLUSTRATOR, AND ADOBE PHOTOSHOP

Something new to designing with Flex is the use of a graphical skin. A skin is a graphical treatment that is applied to a user interface component to give it a new look. Skins come in two basic varieties—graphical and programmatic. Using CSS, a key-value relationship is established between a component state and a specific skin that visually represents that state. A component can have multiple states—up, down, over, disabled, etc. Thus, a different skin can be applied for each state. Programmatic skins work in the same way, except that the graphics are created programmatically in ActionScript, using calls to drawing functions. An ActionScript class file contains code for determining which skin should be drawn for a component, based on the component's state. Using the drawing functions can be unwieldy for complex graphics, so programmatic skins are best suited for simple skins. In fact, the default theme for Flex applications, "Halo," is based almost completely on programmatic skins.

For creating skins, Adobe Illustrator, Flash, and Photoshop are most helpful. Flex skin templates are available to assist with skinning components within each of these tools.

Initially we worked within Photoshop, since we had more internal experience using it. We did not feel, however, that it produced graphics as clean and reusable as those created in Illustrator. Illustrator starts with vector graphics that can be resized and manipulated more easily.

For much of our design work, we work directly from Illustrator templates to create the initial design images for our skins. Color and styling are done within Illustrator, and then these images are copied into Flash to be layered and labeled as symbols or instances. Each instance represents a state such as *up*, *over*, *down* or *disabled,* which will then be compiled into a Flash SWF file that is referenced in the CSS. Figure 6 shows the basic Illustrator skin template for the ScrollBar component, which includes a unique skin for each state.

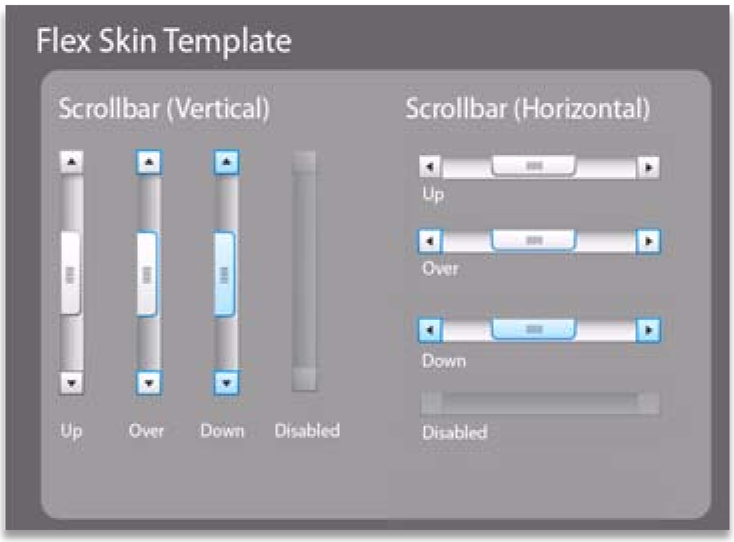


Figure 6. Basic ScrollBar Skin Template from Illustrator

Figure 7 is a snapshot of the Corporate skins for the ScrollBar component. You can see that the basic template has been modified to reflect the Corporate theme color palette.

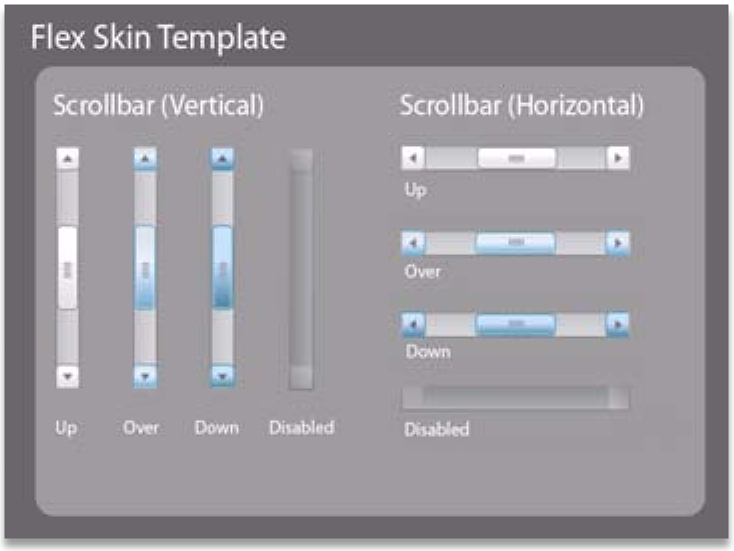
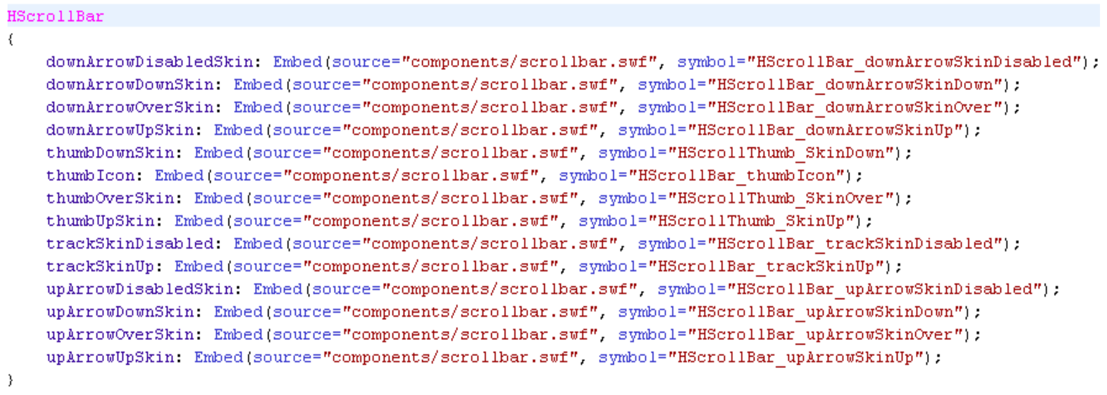


Figure 7. Corporate ScrollBar Skins

The CSS that represents Corporate theme styling for the HScrollBar component is shown in Figure 8. The SWF file that contains the graphical skins is referenced within the CSS, using `Embed` statements for the various states of the HScrollBar component.



```
HScrollBar
{
    downArrowDisabledSkin: Embed(source="components/scrollbar.swf", symbol="HScrollBar_downArrowSkinDisabled");
    downArrowDownSkin: Embed(source="components/scrollbar.swf", symbol="HScrollBar_downArrowSkinDown");
    downArrowOverSkin: Embed(source="components/scrollbar.swf", symbol="HScrollBar_downArrowSkinOver");
    downArrowUpSkin: Embed(source="components/scrollbar.swf", symbol="HScrollBar_downArrowSkinUp");
    thumbDownSkin: Embed(source="components/scrollbar.swf", symbol="HScrollThumb_SkinDown");
    thumbIcon: Embed(source="components/scrollbar.swf", symbol="HScrollBar_thumbIcon");
    thumbOverSkin: Embed(source="components/scrollbar.swf", symbol="HScrollThumb_SkinOver");
    thumbUpSkin: Embed(source="components/scrollbar.swf", symbol="HScrollThumb_SkinUp");
    trackSkinDisabled: Embed(source="components/scrollbar.swf", symbol="HScrollBar_trackSkinDisabled");
    trackSkinUp: Embed(source="components/scrollbar.swf", symbol="HScrollBar_trackSkinUp");
    upArrowDisabledSkin: Embed(source="components/scrollbar.swf", symbol="HScrollBar_upArrowSkinDisabled");
    upArrowDownSkin: Embed(source="components/scrollbar.swf", symbol="HScrollBar_upArrowSkinDown");
    upArrowOverSkin: Embed(source="components/scrollbar.swf", symbol="HScrollBar_upArrowSkinOver");
    upArrowUpSkin: Embed(source="components/scrollbar.swf", symbol="HScrollBar_upArrowSkinUp");
}
```

Figure 8. Example of HScrollBar CSS

In addition to the software tools that we describe above, we also found the Flex SDK documentation useful. This documentation provides information about which style properties each base Flex component supports, as well as general information about style support in Flex.

DEVELOPMENT OF THE SAS CORPORATE THEME

Prior to this effort, SAS's development focus has been to deliver proven solutions that drive innovation and improve performance. Now we want to go one step further and provide great products that look and feel as if they have all come from one great place, which is SAS. The mission of the SAS Flex Themes project was to create a SAS Corporate theme that provides a unique, yet standard, look and feel for all SAS applications developed with Flex. This standard design is the default for all applications to use as we continue to build a presence in the market as one large suite of SAS products.

DESIGN

The Corporate theme was designed with SAS's visual branding in mind. This includes printed materials (Figure 9) and our external Web site (Figure 10), for example.

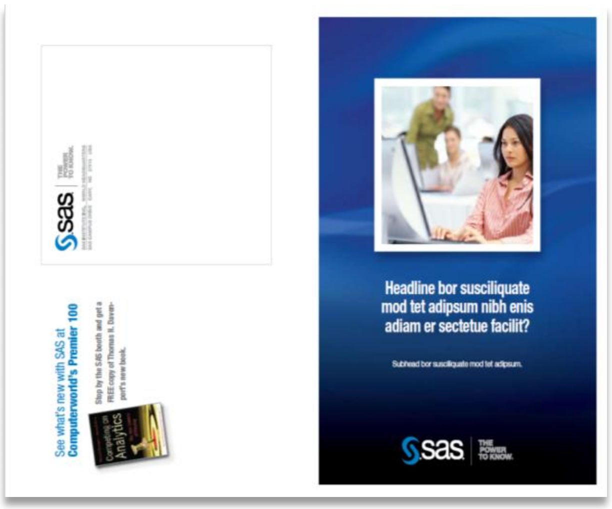
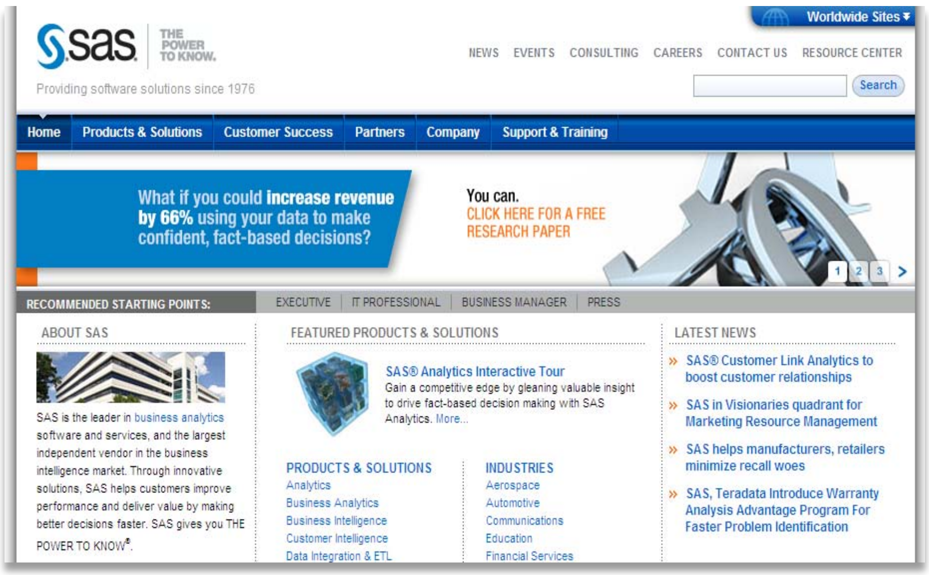

See what's new with SAS at Computerworld's Premier 100
Stop by the SAS booth and get a FREE copy of Thomas H. Davenport's new book.
Competing on Analytics
Headline bor susciliquate mod tet adipsum nibh enis adiam er sectetue facilit?
Subhead bor susciliquate mod tet adipsum.
sas THE POWER TO KNOW.

Figure 9. A SAS Bi-fold Mailer



Providing software solutions since 1976

What if you could increase revenue by 66% using your data to make confident, fact-based decisions?

You can. CLICK HERE FOR A FREE RESEARCH PAPER

ABOUT SAS

SAS is the leader in business analytics software and services, and the largest independent vendor in the business intelligence market. Through innovative solutions, SAS helps customers improve performance and deliver value by making better decisions faster. SAS gives you THE POWER TO KNOW®.

FEATURED PRODUCTS & SOLUTIONS

SAS® Analytics Interactive Tour
Gain a competitive edge by gleaning valuable insight to drive fact-based decision making with SAS Analytics. More...

PRODUCTS & SOLUTIONS
Analytics
Business Analytics
Business Intelligence
Customer Intelligence
Data Integration & ETL

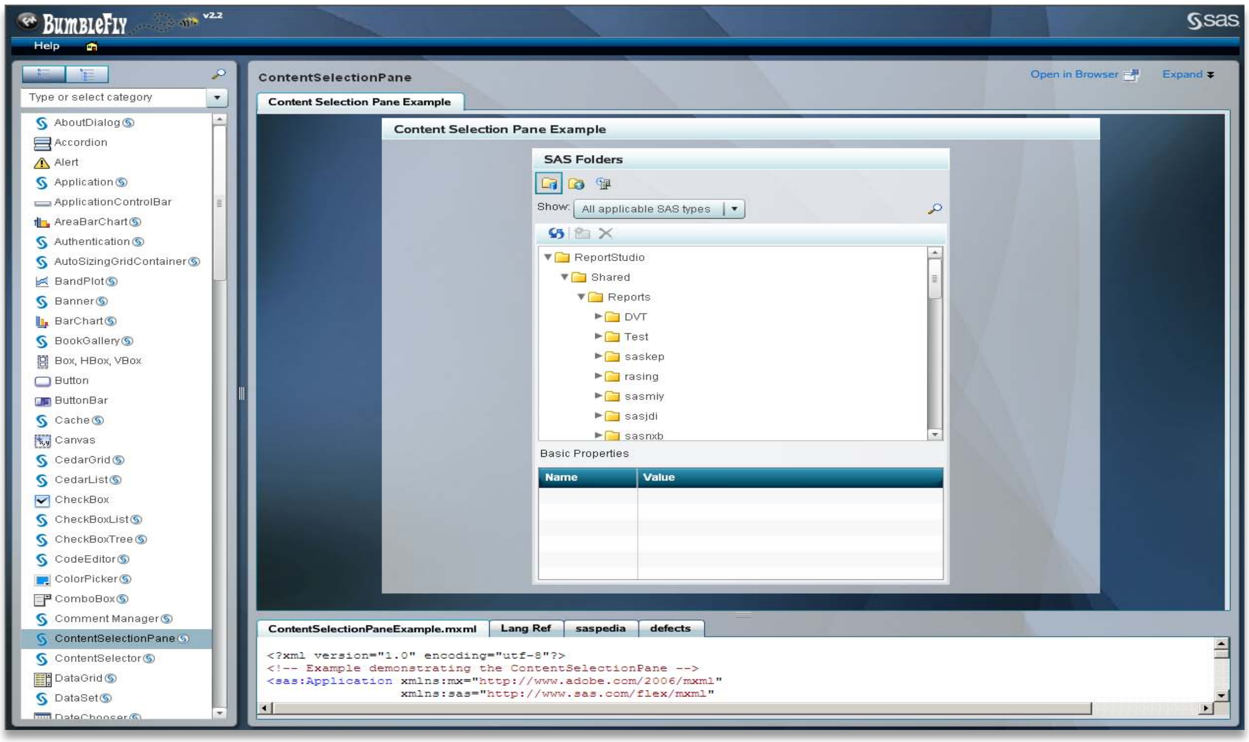
INDUSTRIES
Aerospace
Automotive
Communications
Education
Financial Services

LATEST NEWS
» SAS® Customer Link Analytics to boost customer relationships
» SAS in Visionaries quadrant for Marketing Resource Management
» SAS helps manufacturers, retailers minimize recall woes
» SAS, Teradata Introduce Warranty Analysis Advantage Program For Faster Problem Identification

Figure 10. SAS External Web Site

Due to their design, these marketing resources deliver a crisp and clean message to our customers. In designing the Corporate theme, we wanted to continue to use a similar color palette but try to design a theme that had more graphical and multi-dimensional design. We were able to achieve this by using some of the basic concepts inspired by Halo, but pushing the design to provide more of a “Flexy” or “Flashy” feel. For instance, Halo uses transparent containers that, as they are layered, appear to get lighter in color, which in turn visually provides more depth to the look of the containers and to the design as a whole. We adopted this within our new design to help eliminate some issues we had found, such as no definition around borders, blending of containers into each other, and a flatter overall design. We also decided to use a darker version of the SAS branding color palette, and we incorporated a wider range of cool color tones, whereas in previous versions we had stayed closer to just blue, gray and white.

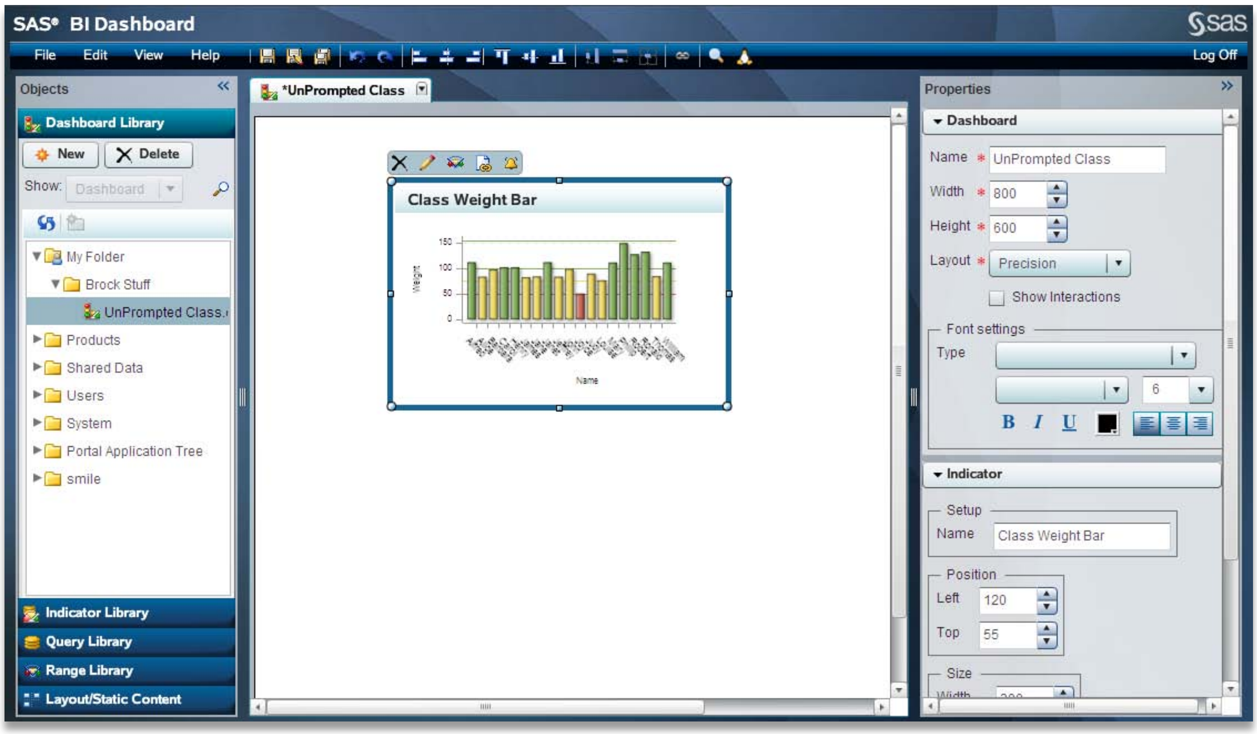
In Figure 11 you can see an example of the layering effect that we achieve within the new Corporate theme by stacking the semi-transparent containers. Additionally, you can see how nicely the light-colored containers provide a contrasting look to the dark background graphic and to the deeper, rich colors that are used on components such as buttons, table headings and within trees. An added boost is provided through the creation of new graphical skins.



```
<?xml version="1.0" encoding="utf-8"?>
<!-- Example demonstrating the ContentSelectionPane -->
<sas:Application xmlns:mx="http://www.adobe.com/2006/mxml"
                 xmlns:sas="http://www.sas.com/flex/mxml"
```

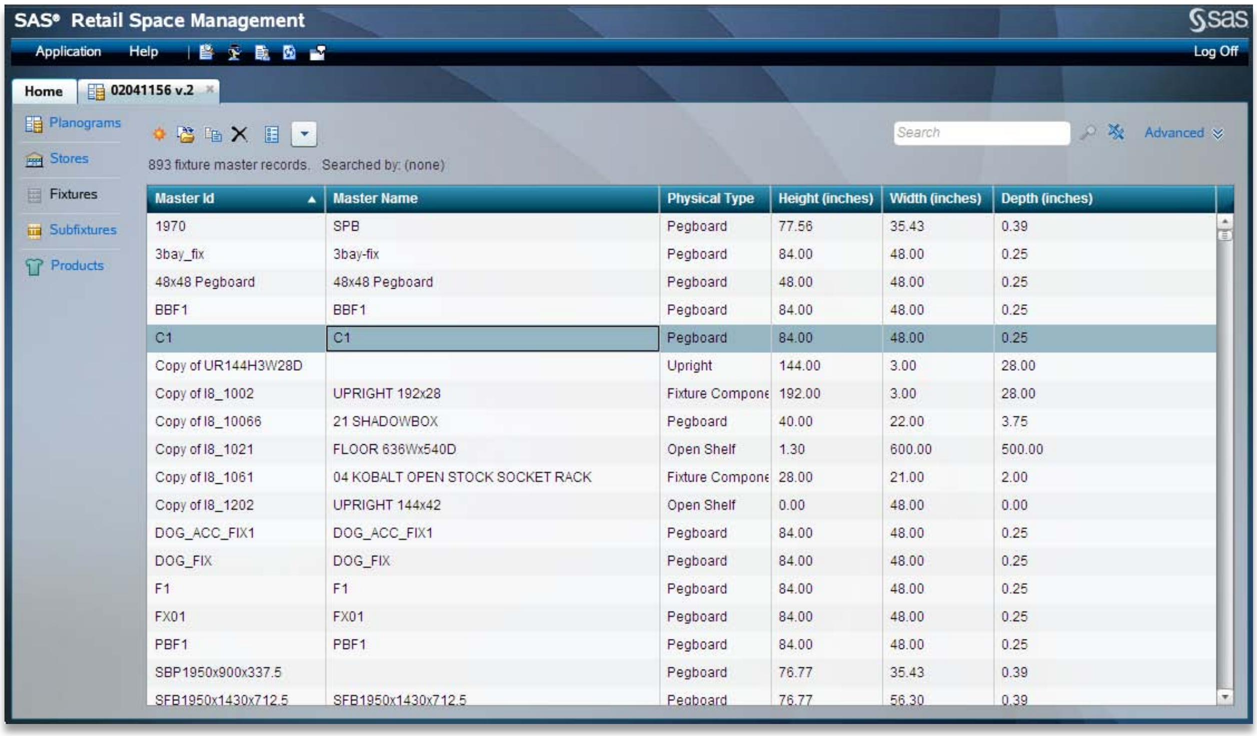
Figure 11. Internal Development Tool That Provides Sample Code and Documentation to SAS Flex Developers. The Flex Themes team also used it to test changes to the Corporate theme. This sample shows container layering styling in the Corporate theme.

The following screen captures (Figure 12 and Figure 13) are of actual SAS Flex applications that use the latest Corporate theme. Seeing the theme applied to different applications shows various components being used together, which helps give a better understanding of the overall look and feel. We used several applications such as SAS BI Dashboard and SAS Retail Space Management as real-life test scenarios for verifying design choices and styling within the Corporate theme.



Name: UnPrompted Class
Width: 800
Height: 600
Layout: Precision
Show Interactions
Indicator Name: Class Weight Bar
Left: 120
Top: 55

Figure 12. SAS BI Dashboard



893 fixture master records. Searched by: (none)

Master Id	Master Name	Physical Type	Height (inches)	Width (inches)	Depth (inches)
1970	SPB	Pegboard	77.56	35.43	0.39
3bay_fix	3bay-fix	Pegboard	84.00	48.00	0.25
48x48 Pegboard	48x48 Pegboard	Pegboard	48.00	48.00	0.25
BBF1	BBF1	Pegboard	84.00	48.00	0.25
C1	C1	Pegboard	84.00	48.00	0.25
Copy of UR144H3W28D		Upright	144.00	3.00	28.00
Copy of I8_1002	UPRIGHT 192x28	Fixture Compone	192.00	3.00	28.00
Copy of I8_10066	21 SHADOWBOX	Pegboard	40.00	22.00	3.75
Copy of I8_1021	FLOOR 636Wx540D	Open Shelf	1.30	600.00	500.00
Copy of I8_1061	04 KOBALT OPEN STOCK SOCKET RACK	Fixture Compone	28.00	21.00	2.00
Copy of I8_1202	UPRIGHT 144x42	Open Shelf	0.00	48.00	0.00
DOG_ACC_FIX1	DOG_ACC_FIX1	Pegboard	84.00	48.00	0.25
DOG_FIX	DOG_FIX	Pegboard	84.00	48.00	0.25
F1	F1	Pegboard	84.00	48.00	0.25
FX01	FX01	Pegboard	84.00	48.00	0.25
PBF1	PBF1	Pegboard	84.00	48.00	0.25
SBP1950x900x337.5		Pegboard	76.77	35.43	0.39
SFB1950x1430x712.5	SFB1950x1430x712.5	Pegboard	76.77	56.30	0.39

Figure 13. SAS Retail Space Management

Once the Corporate theme design had begun to settle, we tested and reviewed it within each SAS Flex application and several mockup interfaces to see how it looked overall. Some of the items we considered during the review process included ensuring that each piece of an application's user interface worked well with its container; making sure that the spacing was correct between the Banner component and the main application area; and testing that the mouse hover colors (on components that support them) work with the overall design. We found some issues. One example is that a neutral color that was used for the up state on CheckBoxes and RadioButtons made them appear as if they were inactive. We addressed this issue by specifying more color for these components.

IMPLEMENTATION

Having already had experience in building themes for traditional JSP/HTML-based Web applications, we were already familiar with what we could achieve using CSS and icons. As far as the actual CSS implementation, we agreed that primarily specifying styles via type selectors would be the best strategy. This is how the Halo styles are implemented: Each component has a corresponding type selector that contains styles to apply to all instances of that component. Here is a snippet of the Halo type selector for the Panel container:

```
Panel
{
   borderColor: #E2E2E2;
   cornerRadius: 4;
   dropShadowEnabled: true;
}
```

The SAS Corporate theme CSS also includes a Panel type selector to override certain Halo styles for Panel, a subset of which is shown below:

```
Panel
{
   borderColor: #333333;
   cornerRadius: 0;
   dropShadowEnabled: false;
}
```

Flex also provides support for a `global` selector. This selector contains any styles that should serve as global defaults across all components. The Halo theme specifies most of its styles in the `global` selector, with any component-specific customizations handled by type selectors. The SAS Corporate theme also includes global styles, such as the *themeColor* style, to override some of the Halo defaults.

Another type of selector that Flex supports is the class selector. One can add a specific class selector to Flex CSS that defines a set of particular styles. For component instances that require the styling represented by that class selector, the selector can be associated to the component instance by setting the *styleName* property on the component to the name of the class selector. In the following example, the class selector *.important* could be associated with any text components that support the *fontWeight* and *color* styles in an application. A developer could set a Label instance's *styleName* property to "important" to apply this styling.

```
.important
{
   fontWeight: "bold";
   color: #FF0000;
}
```

To facilitate the implementation of cutting-edge Flex user interface designs, SAS has created its own custom Flex components. While most of these custom components are composites of base Flex components and thus adopt base styles, some custom components must be styled beyond the base styles. Styling for these custom components is also handled by type selectors that correspond to the component's class name.

Along with the use of CSS and icons, we also wanted to take advantage of something new that Flex brings to the table, which was the use of "skins". When deciding how to implement the Flex Corporate theme, we considered using mainly graphical skins to create the custom look. We agreed that the best plan was to use graphical skins where they provided the most visual value, and to use CSS for the rest of the look. This allowed us to take advantage of Flex's style inheritance. Halo styles and programmatic skins are loaded as a base for all Flex applications if a theme is not specified at compile time, which is the case for SAS Flex applications. Thus, the Corporate theme CSS consists of styles that override the Halo styles.

In spite of the visual design advantages that graphical skins have, there are some disadvantages to using mainly graphical skins for a Flex theme. One is that loading several small graphics for a complex user interface can affect the overall performance of the application. Each graphic uses some memory, and if there are a large number of components that compose a particular user interface, that memory usage adds up. Another disadvantage of graphical skins is that they are not as flexible as programmatic skins. Because the colors in a graphical skin are hardcoded, they cannot be easily customized. A new graphic must be created if a different color variation is desired. Programmatic skins do offer flexibility, since their colors are usually specified using style properties.

A LOOK INSIDE THE SAS FLEX THEMES INFRASTRUCTURE

For our purposes, a theme is a set of one or more stylesheets that are compiled into SWFs so that they can be loaded at run time. These are known as style modules. An XML file defines the themes that are available to applications, and also defines which style modules must be loaded for each theme and in what order. Flex also supports specifying a stylesheet at compile time, thus “baking it in” to the application. We decided against this option because it does not lend well to maintenance or modularity.

We have implemented a theme hierarchy that allows common styles to be defined at a base layer. Specialized styles are defined as deltas to the base set of styles. Figure 14 illustrates the hierarchy that we have created for the current set of Flex themes. The Flex default theme, Halo, is always loaded initially. We created an “abstract” BaseTheme to define any styles that are common across any application theme. Styles in BaseTheme are mainly for custom components or to define usability standards (for example, the design of the Log Off button within the Banner). Most font styles and the ImageLibrary (discussed later) are also defined in BaseTheme. The Corporate theme extends BaseTheme by defining delta styles to either the BaseTheme styles or Halo styles.

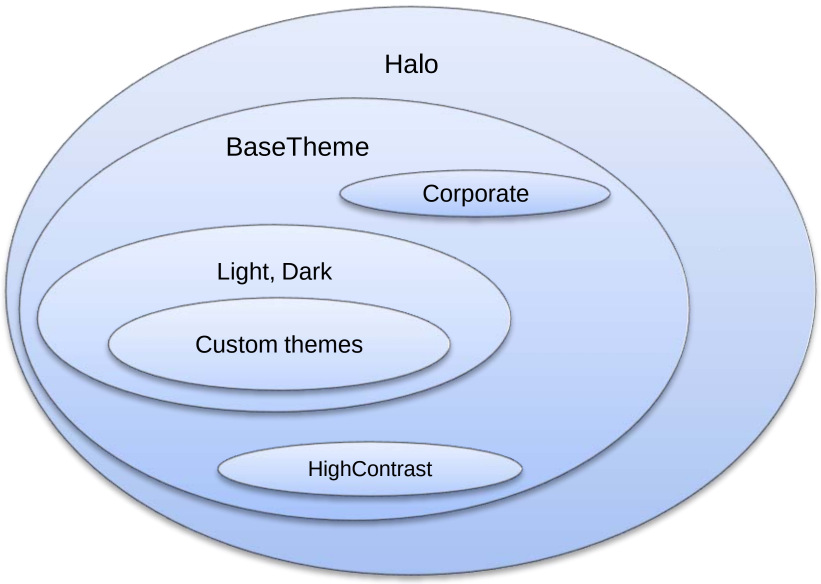


Figure 14. SAS Flex Theme Hierarchy

THE IMAGE LIBRARY

The ImageLibrary is a concept that we defined to help organize Flex icons and images within themes. Previously, with Web themes, the set of icons grew over time into an unmanageable, massive set of files. The nature of the Web theme architecture required these icons to be duplicated for each theme. When work began on the Flex implementation, we seized the opportunity to make a fresh start with icons in themes.

The ImageLibrary consists of an “ImageLibrary” type selector in CSS and an ActionScript utility class to provide easy access to the images. Instead of individual GIF, PNG, or BMP image files, Flex images are created as vector graphics in Adobe Flash or Adobe Illustrator. Vector graphics have benefits over bitmap graphics in that they scale well and have a smaller file size. Using vector graphics allows a designer to create a singular image that can be sized programmatically for different uses. The disabled or inactive appearance of an icon can also be handled programmatically. These factors prevent the need to create and maintain multiple instances of an image at different sizes and states.

Another benefit of the ImageLibrary involves the CSS implementation. The ImageLibrary selector simply contains style properties that point to embedded graphics in the SWF file that contains all Flex images. Any theme that extends the BaseTheme can provide a customized image for a particular usage simply by overriding the corresponding style property. For example, if the BaseTheme defines a standard Delete icon that is black, the HighContrast theme CSS can override just that icon to provide a bright green Delete icon. This is a vast improvement over the Web implementation of themes, which required unique image files for normal icons in addition to disabled versions of the same icons. Additionally, image files were required to be duplicated for each unique theme. With Flex themes, only the handful of images that are unique from theme to theme need to be explicitly overridden. The Flex themes implementation for image handling saves time and effort not only for the team that handles icon development and maintenance, but also for the team that maintains the Flex themes themselves. The ImageLibrary will also prove to be beneficial to applications in knowing, maintaining, and staying fresh with their icons as they move on to subsequent releases.

LOADING STYLE MODULES

Some of the early discussions about the Flex themes infrastructure design involved whether or not to load stylesheets at run time or to compile a stylesheet into an application. Flex supports both methods, but loading stylesheets at run time is better suited to the modular design of SAS Flex themes. In this way, stylesheets loaded at run time, or style modules, can be loaded in a particular order, thus creating style inheritance.

In order for a Flex stylesheet to be loaded at run time, it must be compiled into a style module. This is simply a CSS file that is compiled like a Flex application, so the resulting file has a .SWF extension. The Flex themes implementation relies on a configuration file that specifies which style modules should be loaded for a theme and in what order. Also in play are locale-specific style modules that define any locale-specific styles. These styles mainly include ImageLibrary overrides for images that vary per locale (such as the common *B*, *I*, and *U* icons for Bold, Italic, and Underline text style buttons) and also embedded fonts for languages with specific character sets (particularly for the Chinese, Japanese, and Korean locales).

Figure 15 provides a high-level illustration of how the theme loading process works for the Corporate theme. Note that the locale-specific CSS is loaded only for non-English locales. Not all locales require additional styles or images.

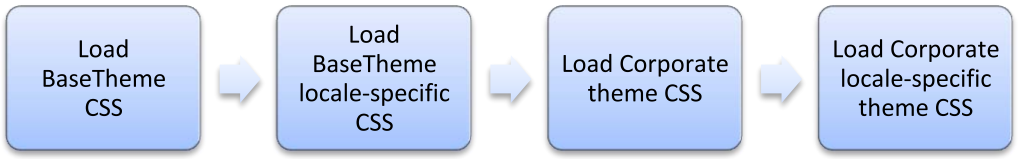

Load BaseTheme CSS
Load BaseTheme locale-specific CSS
Load Corporate theme CSS
Load Corporate locale-specific theme CSS

Figure 15. Theme Loading Process for the Corporate Theme

The theme configuration file also includes information that is required for optimized loading. As development of Flex themes progressed and more applications have come on board, more style modules have been added with each internal release of Flex themes. Because each style module that is loaded at run time uses a portion of memory and increases application start-up time, optimization became a priority. In the configuration file, the style modules are associated with the particular products or libraries that require them. When an application starts up, the optimization algorithm ensures that only the minimum set of style modules are loaded for the current theme and for the particular application.

TECHNICAL CHALLENGES

A major challenge that the SAS Flex Themes team faced while developing the SAS Corporate theme is that Flex CSS is not completely equivalent to the standard CSS that most designers and developers are familiar with. While Flex CSS does adopt some syntax, a limited set of style properties, and inheritance concepts from standard CSS, there are many differences. Here are a few:

- Cannot apply more than one style class to a component instance.
- Child selectors are not supported.
- Commonly used properties such as "background-repeat" are not supported without third-party libraries.

When working with Flex styles, it is important to understand the style inheritance hierarchy. There is an order of precedence for style properties that can trip up even the most experienced Flex developer. From first to last, the order of precedence is:

- Inline (setting a style directly in the MXML for a component or container).

- Class selector.
- Type selectors (the most immediate class takes precedence when multiple selectors apply the same style property).
- Ancestor class's type selector.
- Parent chain.
- global selector.

If a component instance calls *setStyle()*, it takes precedence over all styles, including inline styles.

One lesson learned from the SAS Web theme implementation was not to rely too heavily on class selectors. The default Web theme provides several class selectors that consumers can reference in their code. Unfortunately, over time, Web themes suffered from "selector bloat": With each new release, new components or bug fixes required the creation of new selectors. Some selectors were poorly named, so maintenance was impossible. With the SAS Flex themes implementation, we realized that class selectors were necessary for some scenarios, such as styling a piece of a custom component. The general policy for Flex themes is not to create general purpose class selectors unless necessary.

A seemingly positive feature of Flex is its support for embedded fonts. Embedded fonts are just that: font information embedded into a Flex application or style module. For example, you can embed the Arial font in a Flex application and set all text components to use it. A major benefit of using embedded fonts for text in a user interface is that they produce a smooth, anti-aliased appearance. Additionally, embedding a font guarantees that it is available on the user's system. This is an improvement over standard HTML-based Web applications, which relied on fonts being installed on a user's system. Before deciding to use embedded fonts, it is important to understand the implications.

- Embedding a font increases the resulting SWF size. Embedding a font essentially incorporates that font with the rest of an application's assets, including compiled MXML and ActionScript code, images, and so on. Some fonts are very large to begin with. This issue can be addressed by specifying Unicode character ranges in the embed directive. This allows only a particular set of characters to be embedded instead of the entire set. If your application will display only a known set of characters, then specifying a Unicode character range to embed is recommended.
- Internationalization can present a challenge when multiple character sets are in play. Consider this scenario: A Flex application uses an embedded font to display text. The font contains only Latin characters and punctuation. The user's locale is set to French, but the Flex application attempts to display text containing Chinese characters. The embedded font does not contain the Chinese characters. Instead of falling back to a system font that does contain the Chinese characters, Flex simply does not display any character. Or, for some fonts, empty rectangles are displayed to indicate the missing characters. This can negatively impact data integrity and accuracy, since a user viewing a data table would not necessarily know that data was missing as opposed to data not being available.
- Flex documentation acknowledges that its component measurement algorithms have trouble with embedded fonts. When certain components that use embedded fonts are displayed, text can sometimes be truncated or overlap other components. The documentation recommends adding padding in these cases.

DESIGN CHALLENGES

The main design challenge that we faced and will continue to face is creating themes that look good and that "just work" across all SAS Flex applications despite their different implementations. Outside of SAS, Flex applications are usually created with one theme in mind, since they are usually a single product without a diverse set of users. This scenario almost eliminates the challenges that we have experienced with Flex theme development at SAS.

One major issue that we experienced while developing the Corporate theme was that although individual components by themselves looked and worked great in a theme, when the components were put together in actual applications, visual indicators such as container edges and visual depth within the design were lost. Layering a Panel container within a larger Panel caused the two Panels to blend together, since they were colored the same (Figure 16).

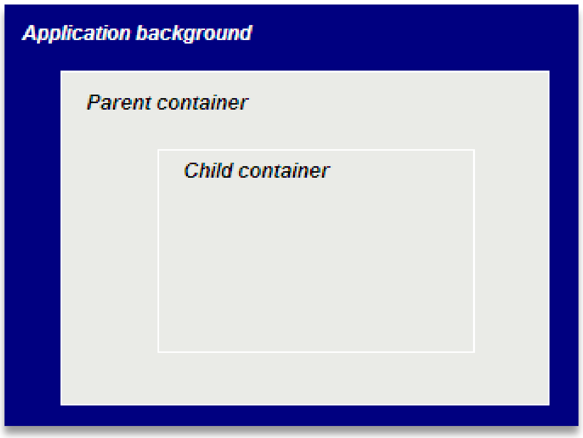


Figure 16. Example of Container Layering Issue When Using One Flat Color

We experimented with some ideas to fix this issue, such as setting a more prominent border color, but that tended to create a flat “coloring book” appearance. We also considered creating additional versions of each Flex container with either lighter or darker color values or different graphic styles, depending on the container’s usage. The intent was to provide application developers with a primary container, an optional secondary container, and possibly a third-level container to be used as needed. Eventually we decided that it would not only be a large task to define, maintain, and document for consumers, but that it left too much of the theme design open to interpretation, compromising our goal of having a standard look and feel for all SAS applications. In the end, we went back to the basics by revisiting the Halo theme and concluded that the key to its working so well for almost every application is the use of its slightly transparent containers.

Another challenge that we encountered was working around bugs in the Flex components themselves. An example is our attempt to style DataGrid column headings. Styling this piece initially seemed simple enough, but it eventually required researching, debugging, and working around a Flex SDK shortcoming. As with most problems that come up unexpectedly, they tend to snowball, resulting in revisiting and adjusting other design elements within the theme to accommodate the solution needed for the original problem.

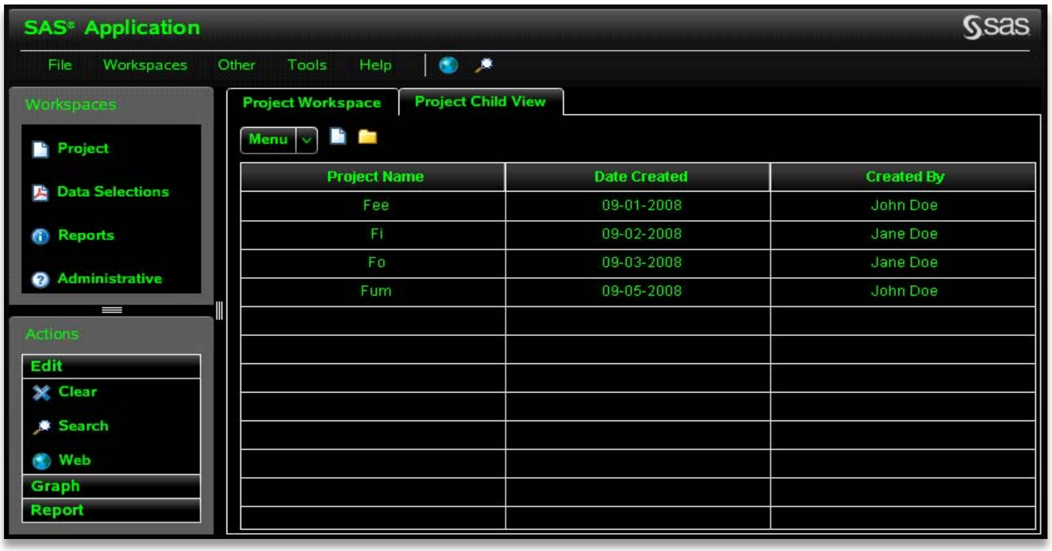
To add to the list of challenges, there was a need to coordinate the appearance of the Corporate Flex theme with the appearance of the default Web theme. These “hybrids” combine both Web-based (HTML/CSS/JavaScript) and Flex user interfaces. One example of a hybrid application is the SAS Information Delivery Portal. The Portal displays the default Web theme, but portlets within the Portal can be implemented in Flex, and can thus use the Corporate Flex theme. This presents a design challenge when considering that the color palettes between the default Web theme and the Corporate Flex theme must coexist harmoniously in one browser window. In the end, users do not need to know that they are using a hybrid application. The transitions between the two sets of technologies should be seamless.

THE FUTURE OF FLEX THEMES

For SAS Web applications, only one Web theme is provided, aptly named “default.” It consists of mainly gray and blue tones. Considering the long development cycle for just this one theme, providing additional Web themes was not an option. On the other hand, Flex, with its built in support for styling, made providing multiple themes a real possibility. In addition to the default Corporate theme, we will be providing three additional themes in a future release: High Contrast, Light, and Dark.

HIGH CONTRAST THEME

The primary intent of the High Contrast theme (Figure 17) is to enable SAS Flex applications to meet the U.S. Government Section 508 standards for accessibility and to provide a useful theme for users who have visual disabilities. The High Contrast look is similar to the Microsoft Windows operating system’s “High Contrast #2” theme, which provides a black background and high visibility foreground colors to present key features of the user interface. This theme was intentionally designed without graphical skins, relying on CSS for some minimal design elements such as gradation of tabs and buttons.

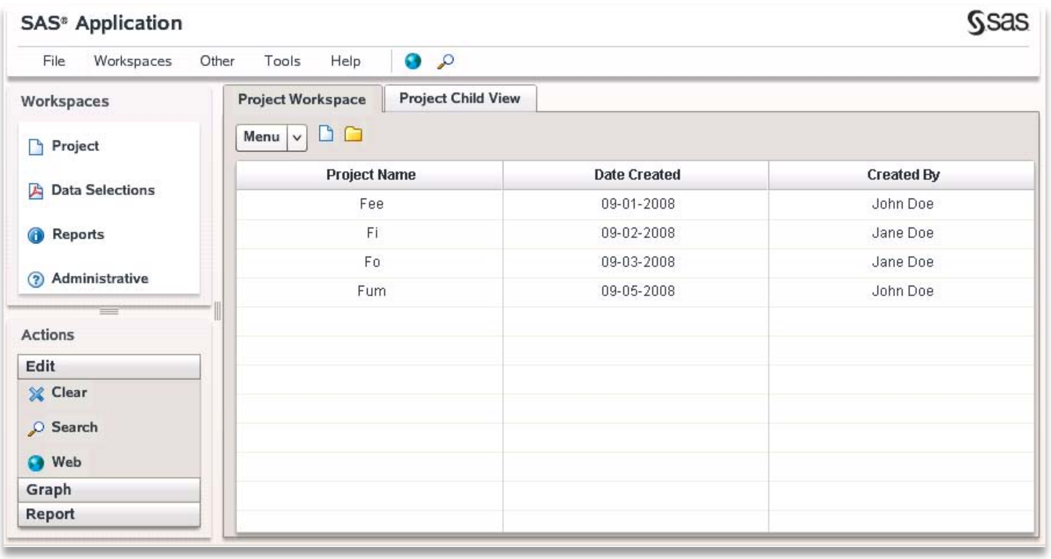


Project Name	Date Created	Created By
Fee	09-01-2008	John Doe
Fi	09-02-2008	Jane Doe
Fo	09-03-2008	Jane Doe
Fum	09-05-2008	John Doe

Figure 17. SAS High Contrast Theme

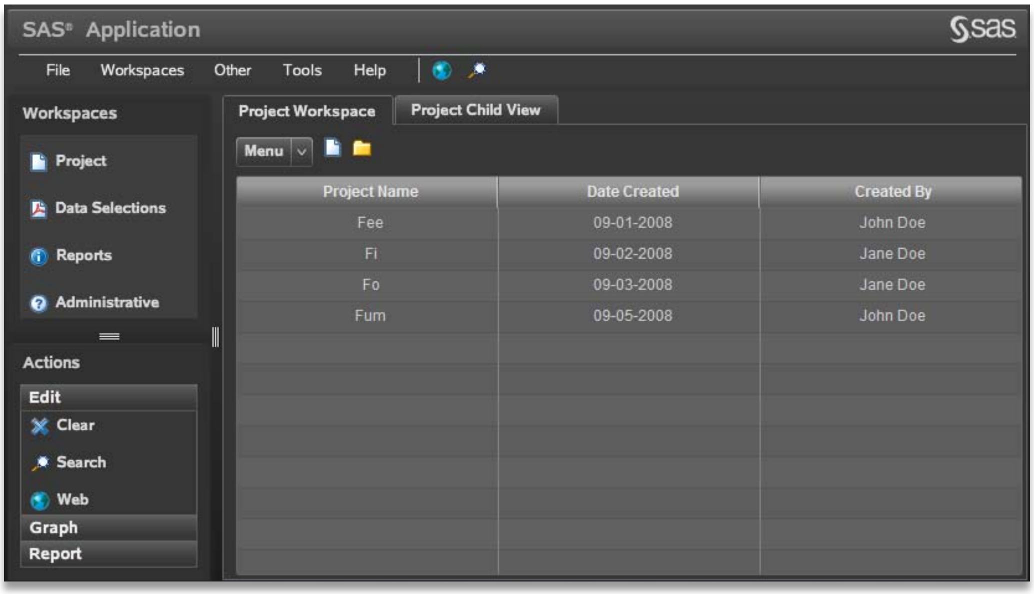
LIGHT AND DARK THEMES

Two additional Flex themes, Light and Dark, were created simply by overriding the Halo styles. The intention was for these themes to be implemented in a minimal way and without custom graphical skins. The Light theme (Figure 18) uses shades of light gray, beige, and white for its color palette. The Dark theme (Figure 19) is based on darker, charcoal gray tones.



Project Name	Date Created	Created By
Fee	09-01-2008	John Doe
Fi	09-02-2008	Jane Doe
Fo	09-03-2008	Jane Doe
Fum	09-05-2008	John Doe

Figure 18. SAS Light Theme



Project Name	Date Created	Created By
Fee	09-01-2008	John Doe
Fi	09-02-2008	Jane Doe
Fo	09-03-2008	Jane Doe
Fum	09-05-2008	John Doe

Figure 19. SAS Dark Theme

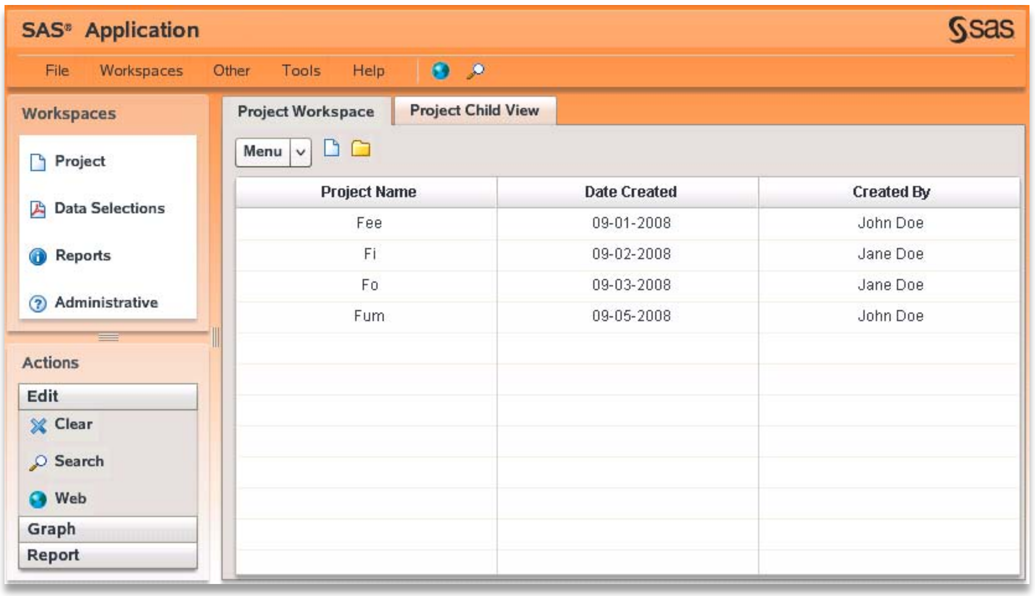
CUSTOM THEMES

The infrastructure for Flex themes will support the concept of custom themes, as did the Web themes infrastructure. A customer might not want to view their SAS applications in the default Corporate theme. They might want to apply their own branding and customizations to help SAS applications fit in with the rest of their software environment. In this case, the customer can create a new custom theme that includes CSS and optionally any images that they want to use in place of the default Corporate styles and images.

Creating and applying a custom Web theme can be a complicated and time-consuming endeavor. The user must first run a special program to generate a new set of theme files. The user then customizes the theme files based on a color palette or from scratch. Then a SAS job must be run to register the new theme in metadata. The Flex themes implementation simplifies the process by providing the Light theme and the Dark theme as bases for creating new custom themes.

To create a custom Flex theme, first choose either the Light theme or Dark theme based on the similarity to the desired color palette. A company whose dominant corporate color is a rich orange might want to begin with the Light theme as a base for their custom theme. The Light theme is based on a warmer palette with yellows and more neutral colors that would complement the orange. The company can then further customize the theme to its branding needs by switching out the SAS logo with their own logo. They might even go as far as adding a graphic to the overall background of the theme, such as in the Corporate theme, thus adding more visual interest to their custom design.

The custom theme CSS contains only the differences from the customer's choice of the base Light or Dark themes, so the resulting CSS would ideally be minimal. Images, icons, and font styling are inherited from the BaseTheme and the Light or Dark theme. An example of a custom theme based on the Light theme is shown in Figure 20. In this example, only the style property `backgroundGradientColors` was overridden to create an orange theme. As you can see, this simple style change provides a quick, yet substantial, visual impact for our customers.



Project Name	Date Created	Created By
Fee	09-01-2008	John Doe
Fi	09-02-2008	Jane Doe
Fo	09-03-2008	Jane Doe
Fum	09-05-2008	John Doe

Figure 20. Example of a Custom Theme Based on the Light Theme

Admittedly, creating a custom theme is not always a straightforward procedure. With Flex we have an opportunity to improve this process. Our goal is to create an easy-to-use application to help customers create simple, visually appealing custom Flex themes. And who knows, maybe you will even see more theme choices down the road!

CONCLUSION

We are excited that SAS will be delivering some of its first Flex-based products and solutions in 2010. And with their introduction, the Corporate theme will be released. We believe that we have achieved the goals that we originally set out to meet for this project, which were:

- to provide a unique SAS visual identity across the next generation of SAS products and solutions
- to build an infrastructure that even far in the future will meet our customers' needs by supporting custom themes were their own corporate branding can be applied.

SAS software products are leading the industry in functionality and capability. Now, we hope to reach another level by ensuring that they are visually appealing and user friendly – not just *sexy*, but *Flexy!*

REFERENCES

Adobe Flex 3 SDK Documentation: http://livedocs.adobe.com/flex/3/

"Adobe Flex" on Wikipedia: http://en.wikipedia.org/wiki/Adobe_Flex

ACKNOWLEDGMENTS

We would like to thank Tammy Gagliano, Corey Benson, Todd Barlow, and Jim Toering for generously offering their time and support throughout the Flex Themes project. Thanks also to the entire Flex development community at SAS for their feedback, support, and patience.

RECOMMENDED READING

Sanchez, J. & McIntosh, A. 2008. *Creating Visual Experiences with Flex 3.0*: Addison-Wesley Professional.

ScaleNine LLC's Web site, http://scalenine.com, is a great source of skins and themes for Flex and AIR.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Amy Dull
SAS Institute, Inc.
SAS Campus Drive
Cary, NC 27513
Tel: (919) 531-1251
Fax: (919) 677-4444
amy.dull@sas.com

Sherry Parisi
SAS Institute, Inc.
SAS Campus Drive
Cary, NC 27513
Tel: (919) 531-2786
Fax: (919) 677-4444
sherry.parisi@sas.com