Sterling Web™

User Interface Architecture Guide

Release 9.0

Last updated in HF1

May 2010

Sterling Commerce
An IBM Company

Contents

Introduction to the Sterling Web User Interface

This topic describes how the Sterling Web user interface (UI) is structured. It discusses the technologies used and how they are applied. It also covers the principles and conventions of user interface (UI) development.

The Sterling Web UI provides mechanisms to adopt multiple styles or look-and-feel, based on web stores. The guide focuses on how to structure the UI to maximize the benefits of such mechanisms.

Note: This guide covers only the “view” part of UI development. The detailed development process of the Sterling Web application is covered in the *Developer Guide*.

This guide provides information about the following topics:

- Overview of the Sterling Web UI
- Display structure
- CSS model
- Conventions and best practices
- Exploring themes
- Performing common theme-related modifications

Overview of the Sterling Web User Interface

Sterling Web is a Web-based application that offers product features to users through the World Wide Web. The user interface (UI) is based mainly on the Struts 2 framework, JSP, and JavaScript, and their recommended practices. Asynchronous JavaScript and XML (AJAX) or Asynchronous HTML and HTTP (AHAH) is leveraged to provide dynamic user experience. CSS is leveraged to influence the layout and its look and feel.

The Sterling Web UI is designed and architected to be flexible in configuration and customization. The design also enables themes, and allows the association of a theme with a Web store. As a result, multiple sets of look and feel, and user experience are offered in parallel.

The focus of this document is to develop views of the presentation tier. The server-side components of the presentation tier, such as Action (controller) and interactions with the business tier (Model) are not in the scope of discussion of this document.

In addition to the display of information, and organization of information, there are three important aspects to a UI display:

- look and feel
- layout
- dynamic behavior

To provide a more flexible and themeable UI, it is essential to separate the information structure from the rules for look and feel, layout and dynamic behavior. In other words, it is crucial to separate the HTML from styling and scripting, and apply styling and scripting rules independently.

In the Sterling Web UI, JSP is the choice of technology for a view. It is used to render HTML content that reflects how information from the back end is organized. CSS is used to control the page layout and the look and feel. JavaScript is used to introduce dynamic behavior for better user experience as well as enhanced visual effect and input validation.

Following is a list of frameworks and UI technologies that the Sterling Web UI is built upon, without the specifics of versions and dependencies:

- JSP
- CSS
- Struts 2
- Ext Js

The separation of responsibility can be summarized as follows:

JSP

- Render HTML content, that is, information and its structure
- Include proper CSS and JavaScript files

JavaScript

- Dynamic behavior such as event handling, dynamic content loading, and so on
- Provision of additional visual effects dynamically

- ✦ Input validation (Note that this is not the focus of this document)

CSS

- ✦ Page layout including size, position of panels, controls, and elements of display
- ✦ Look and feel, fonts, color, background images, and so on

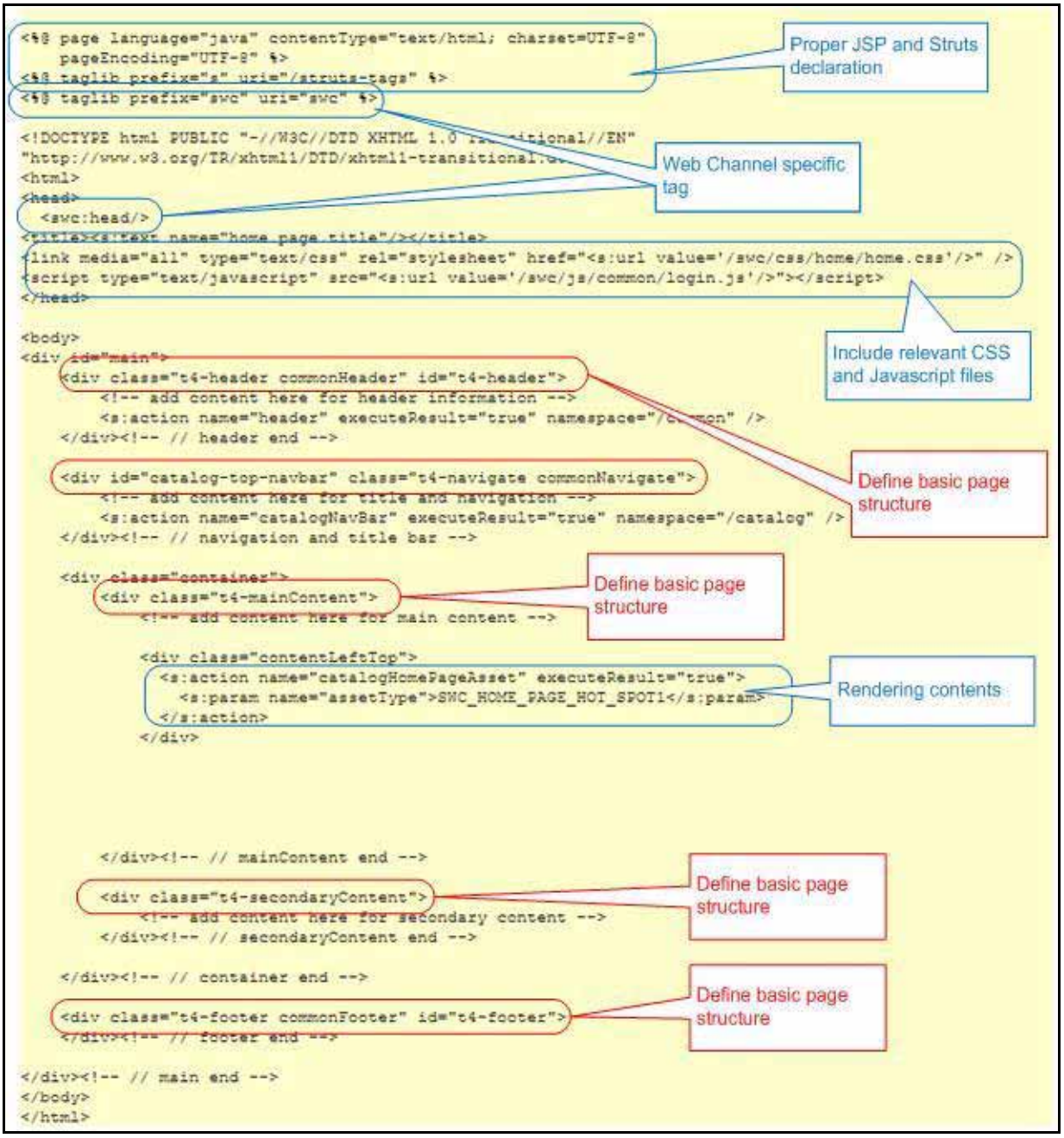
Note: To maximize the benefit of separating content (HTML) from display (CSS) and behavior (JavaScript), the rules for display and behavior for a given page should be captured and externalized as separate files. After that, changing display or behavior for a given page can be achieved by switching the CSS and JavaScript files included.

Understanding JSP

The basic rules that have been established for JSP page development are as follows:

- No inline styling is allowed in JSP.
- CSS classes should be captured in separate files. No embedded class definition is allowed in JSP.
- JavaScript should be kept in separate files, and shared scripts should be captured in common library files.
- Avoid inlining the JavaScript for event handling in the JSP. Encourage the definition of the binding between event-handlers and page elements in separate files so that behavior can be modified without the JSP being changed.

The following diagram displays a typical JSP in Sterling Web.



This figure illustrates the following:

- JSPs in Sterling Web are based on Struts 2.
- Usage of the Sterling Web tag `<swc:xxx>`.
- JSPs include the CSS and JavaScript captured in separate files.

- JSPs define the basic page structure with `<DIV>`.
- JSPs render information.
- JSPs do not embed the `<STYLE>` block.
- JSPs should avoid including the `<SCRIPT>` block, and capture the JavaScript in page-specific files.

Using the <swc:head/> Tag

As illustrated in the previous figure, one important tag used in the JSPs is `<swc:head/>`. Every top-level JSP in Sterling Web UI should define this tag inside the HTML `<head>` section. This tag renders HTML code fragments to include all the basic and shared JavaScript libraries and CSS files, other than those included specifically by each individual page. Because these sets of files may change from release to release, and as may their location, they are encapsulated into one tag. It is essential to include this tag in all the top-level JSPs for convenience and consistency.

Using JavaScript to Enrich User Experience

In Sterling Web, JavaScript enriches user experience with dynamic interaction.

The usage of JavaScript can be categorized into the following categories:

- Dynamic behavior controls
- Event handling
- Dynamic content loading
- Provision for additional visual effects
- Client-side input validation

JavaScript applying SVG

Sterling Web UI utilizes JavaScript to decorate user interface (UI) dynamically with additional visual effects. The technique involves JavaScript dynamically associating HTML elements with CSS styles and background images. These background images are dynamically calculated and generated by a back-end service with Scalable Vector Graphic (SVG) technology.

The following visual effects are used extensively in the Sterling Web UI. These effects are applied based on the corresponding theme so that many of them can be provided in parallel:

- Rounded borders
- Buttons
- Gradient backgrounds
- Other graphics

With this technique, Sterling Web UI has the following benefits:

- Reduces the necessity to create static images
  - No necessity to create new graphics for different color schemes
  - Dynamic calculation of background image sizes
- Improved code readability – No extra HTML constructs required to create visual effects
- Performance aware – Images are created and cached for improving performance

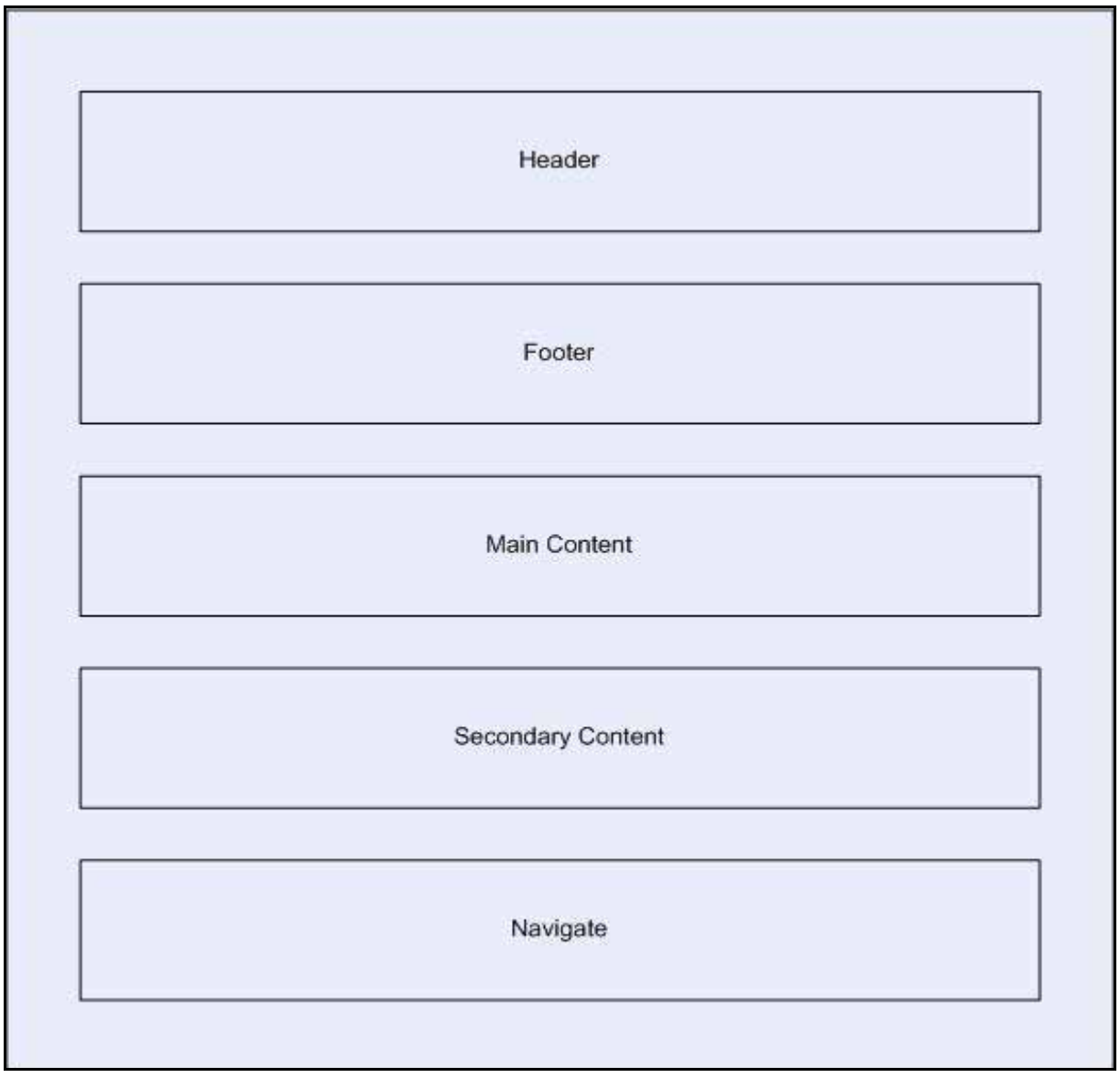
Note: Ensure the JSP includes the `<swc:head/>` tag. The necessary declarations to use the SVG-backed decoration are encapsulated in the `<swc:head/>` tag. The JavaScript functions to decorate the UI are kept in the theme-specific file `~/swc/js/theme/{theme name}/theme.js`. Customization or additional visual effects or both can be added to this file.

Understanding CSS

CSS controls the layout and its look and feel. CSS rules are defined in separate files that are external to the JSP pages. In addition, there are different levels of abstractions among these rules, which are also discussed in detail.

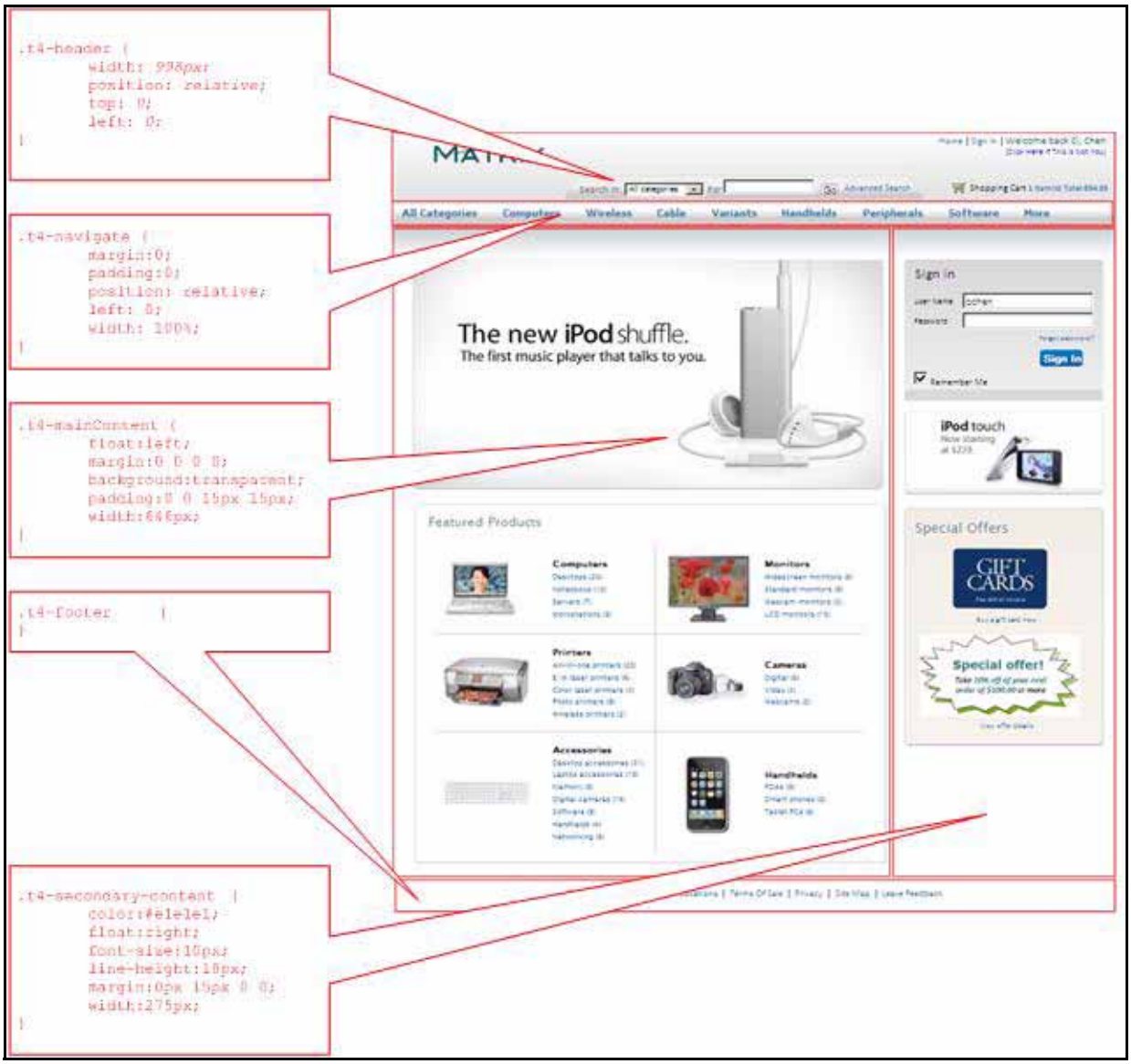
JSPs are responsible for defining a basic page structure. The `<DIV>` tags partition information into multiple blocks.

Without the layout rules defined, a page might look like the following figure:



Layout rules should be expressed through separate CSS classes; inlining or embedding the rules directly in the JSPs is not recommended.

The following figure illustrates how CSS is used to control the layout of top-level panels in a typical Sterling Web application page. These layout rules are shared among JSPs and are grouped as a different layout template. For more information, please refer to the section "CSS Model".



A lot of display factors contribute to the look and feel of a given display, such as the colors and fonts used to display text information, the color, size and shape of UI controls and the content included such as the images, multimedia and so on. Offered as a generic product, the Sterling Web application does not have control over the contents its customers offer. For example, images or video clips are commonly used to attract users' attention as well as convey information and shape product images. However, out-of-the-box

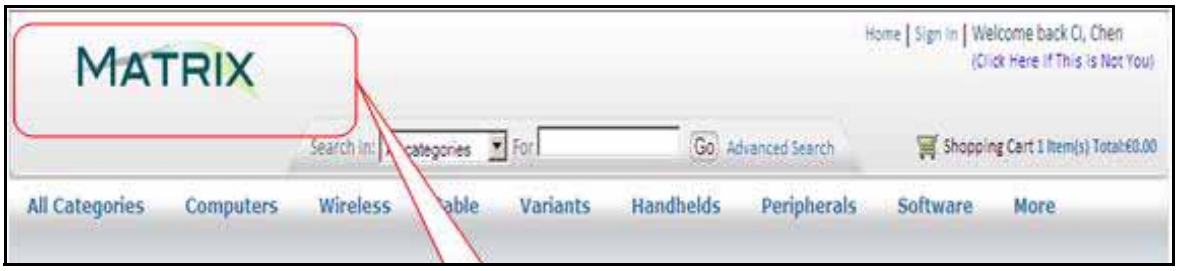
Sterling Web apps cannot rely on these components to build attractive pages. Therefore, the focus is on how to influence the contents, rather than what contents to offer.

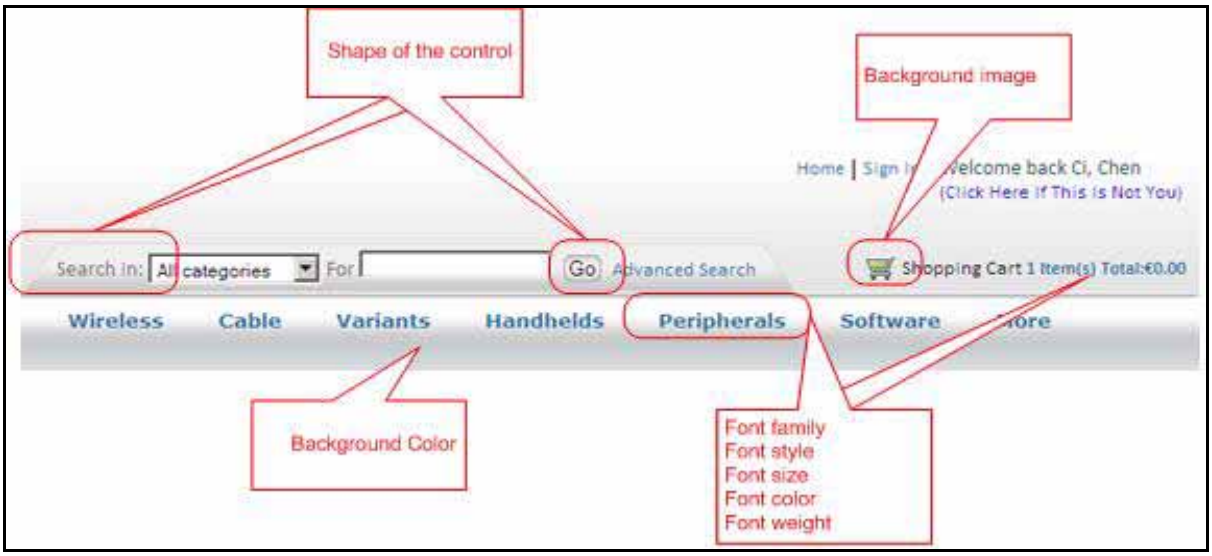
There are three categories of display factors that are mainly controlled through CSS:

- Font
- Color
- Shape (and size) of elements

Note: The third category shape can be achieved by using static images of shape. This can be controlled by CSS by defining background images. However, Sterling Web provides a utility based on SVG that can dynamically shape the elements. It is applied dynamically using JavaScript and CSS in combination.

The following figures illustrate the manner in which display components are controlled through CSS:





CSS Model

There are two system-level requirements that are specifically called out in the Sterling Web UI:

- Template controlled page layout - Pages in which the top-level panels laid out in a similar manner are grouped together. The rules about the position and size of these panels are shared and can be configured globally. Pages with the same rules are said to have the same template. Changing the template definition will affect the layout for all those pages.
- Enable UI theme - The UI should have a consistent look and feel throughout the Sterling Web application. The particular look and feel or theme should be captured, made configurable, and changeable for each Web store. The focus of the display factors are those related to font, color, and background images.

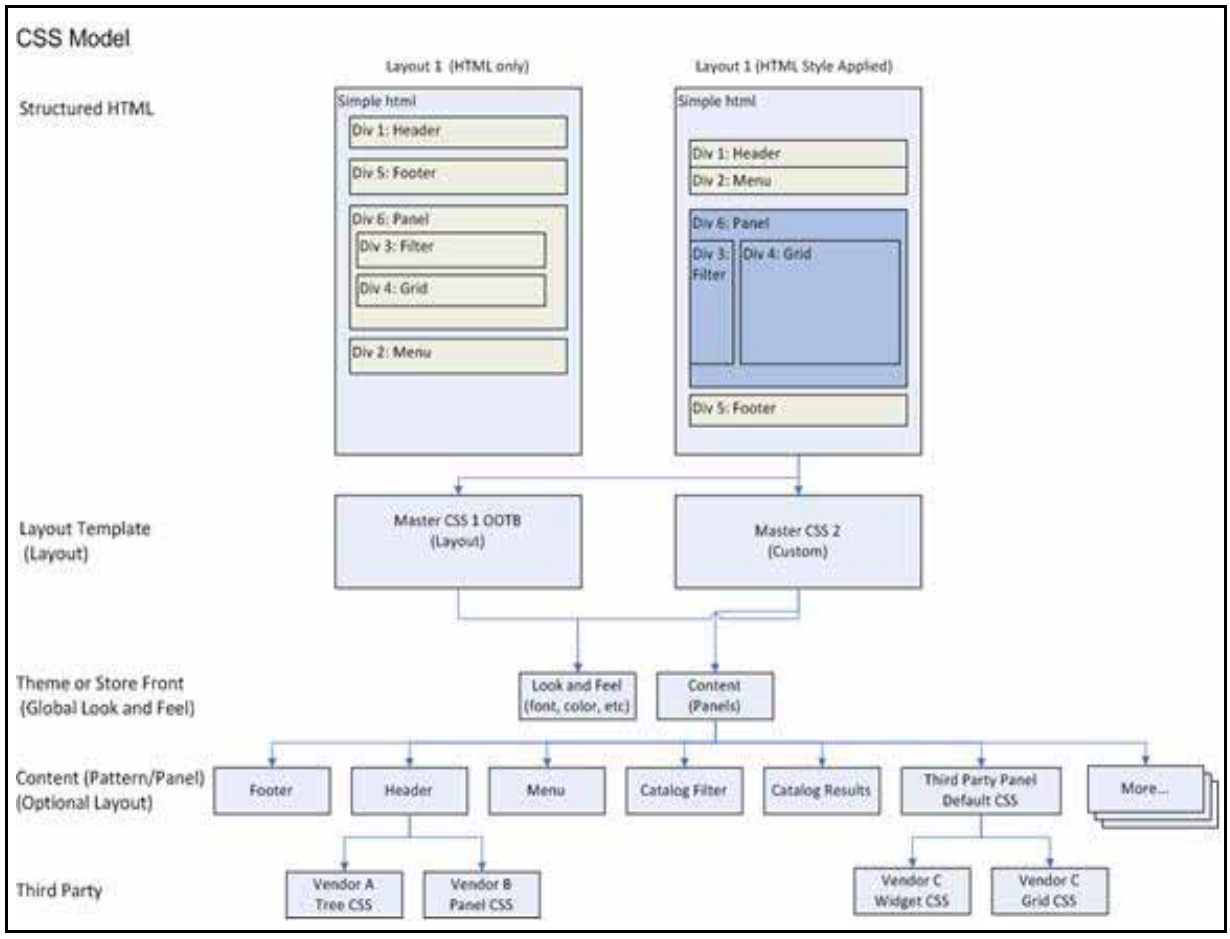
CSS is the main mechanism used to control the UI's look and feel and layout in the Sterling Web application. To provide consistent rules across applications and pages, these rules are shared with different levels of abstraction. Template and Theme are two of these abstractions. Other abstractions have also been introduced to provide more flexibility and better organization.

Levels of CSS

There are four types of CSS used in Sterling Web applications:

- Template - CSS rules specifically defined for the size and position of top-level panels in Web pages
- Theme - CSS rules that constitute a theme, typically those related to font, color, and background images
- Common - CSS rules shared by an application or page
- Page - CSS rules defining display attributes related to a specific page

The following figure illustrates a CSS model.



Template CSS

Template CSS is a shared CSS type that is specifically called to capture the layout of top-level panels in a display. Each top-level JSP in Sterling Web (as opposed to those included by others) is associated with a set of template classes. These classes define the layout rules. The top-level structure expressed through the `<DIV>` tags in the JSP should be assigned to those template classes. The manual step is about how top-level layout rules are enforced.

The `<DIV>` tags in JSPs can be reorganized from unstyled blocks to panels with the desired position and size in the target display.

Note: The original order of the `<DIV>` tags in JSPs is irrelevant. After the JSPs are assigned with a certain template class, the positioning and sizing are completely controlled by the corresponding template rules.

There are different sets of template classes in use. Each set of template classes is captured in a CSS file, with the file name pattern as `template-x.css`, and x enumerated from 1 to n where n can be any numeral. Additional templates can be defined if more types of top-level layouts are identified.

The benefit of enforcing template rules is obvious in configuration and customization. In each deployment, the size of the display can be modified. Users can decide the changes to be made. Since multiple pages share the same CSS, the changes made to one page are reflected dynamically in the remaining pages. The deployed site can decide if the main content (for example, product information) is at the left in relation to the secondary content (promotions), or to the right. Combined with other styling modifications, the site can even decide if the navigation bar should be at the top horizontally, or to the left vertically.

Theme CSS

A theme is a collection of display factors constituting a consistent look and feel throughout displays in applications. In the Sterling Web UI, theme is swappable, and multiple themes can coexist based on different Web stores.

The set of display factors that are considered to be theme related in the Sterling Web UI include:

- Font
- Color
- Background image
- Shape of the UI elements

Except for the last one, all the other factors are captured by styling rules, implemented as the theme CSS, and put into one of the `theme.css` files. Controlling the shape of the UI elements is done through a combination of the theme CSS and JavaScript.

Multiple `theme.css` files can exist in a given deployment of the Sterling Web application. These files typically have the same set of rules or classes, but with different definitions. Sterling Web provides the mechanism to associate different `theme.css` files with a particular Web store. This is done dynamically and automatically, without any programming .

While designing Web pages in the Sterling Web applications, the CSS used should first come from the existing shared CSS. The top-level layout rules should come from one of the `template-x.css` files and the theme-related rules should come from the target `theme.css`. For example, there are different styles of Submit buttons available in theme CSS with implied business meaning, such as main action button, secondary action button, and so on. The developers should carefully choose the matches their application requires. This is how the application maintains a consistent look and feel.

Rules in `theme.css` are sharable, but not all of them are shared. Some of them are used only in one place. However, because they impact the look and feel, and are considered theme related, they are included in `theme.css`. If additional styles are identified during page design, new CSS classes can be implemented to define the rules. If these rules are theme related, they should be promoted from page CSS to theme CSS.

In the topic ”Exploring a Theme”, a set of theme CSS will be examined, and examples will be provided to illustrate what a typical theme CSS looks like.

Common CSS

Shared CSS rules across pages are captured and stored in CSS files in common directories. Other than template and theme-related CSS, these shared rules are further categorized as follows for better organization:

- Global - This CSS file is used to provide default styling rules. It is based on the best practice to ensure consistent default behavior across platforms and browsers. With the explicit default setting, it prevents unintended, browser-dependent, default styles to cascade through. There is only one copy of the global CSS throughout one deployment of the Sterling Web.
- Component - These CSS files are styling rules associated with the UI components used in Sterling Web. These components can come from a third party, and are built in-house, for example, CSS for tab, grid, and so on.
- Common - These CSS files are other shared styling rules. They can come from shared JSP fragments or other application components, for example, CSS for the header and footer JSP fragments.

Page CSS

Each top-level JSP page in Sterling Web is associated with a page-level CSS file, with the exception of the JSPs that have little styling rules, or those to be included by others. It is the pivotal point of how different levels of CSS interact with the JSP.

Two tasks can be typically performed in page-level CSS files:

- Including other CSS files
  - Common CSS - Depending on what page fragments are included or JSP components used, the page-level CSS should include the necessary CSS files.
  - Template CSS - Defining the position and size of the top panels.
  - Theme CSS - Defining the fonts, colors and background images constitute a consistent look and feel.
- Defining local styling rules

  Rather than being embedded in a JSP directly, styling rules that are applicable only to a specific JSP should be defined in the page-level CSS files, for example, the size of a subpanel, column widths of a table, position of text, and so on.

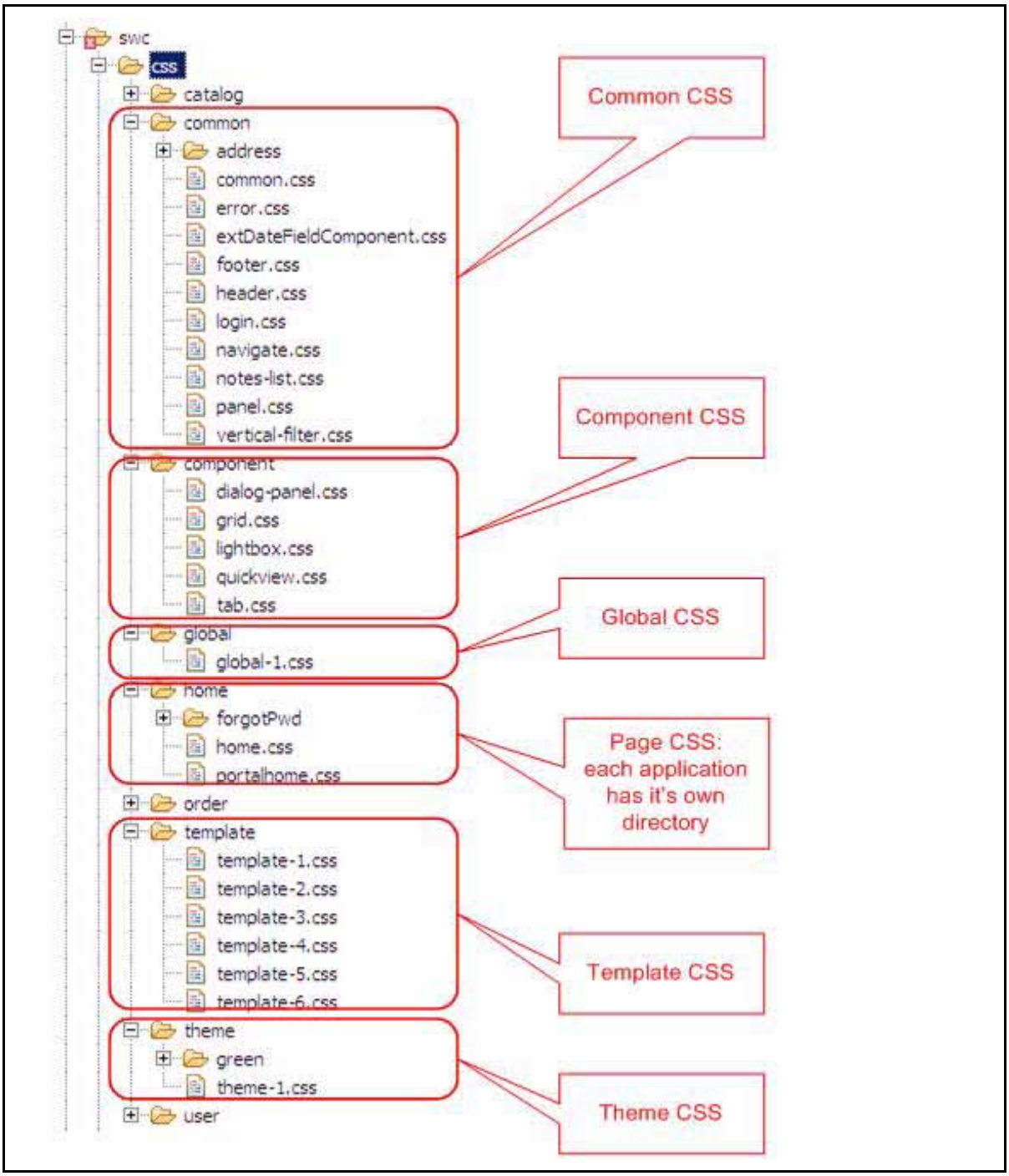
The following page displays a typical page CSS file:

```
@import url(../global/global-1.css);
@import url(../template/template-4.css);
@import url(../common/header.css);
@import url(../common/footer.css);
@import url(../common/navigate.css);
@import url(../component/grid.css);
@import url(../common/login.css);

/*********************************************/
/* content left style*/

#contentLeftBottom, .contentLeftBottom {
  clear:both;
  padding:0;
  margin-left:18px;
  margin-right:auto;
  width: 650px;
}

/*#contentRight {*/
#contentRightTop {
  padding:15px 15px 0 0;
}

#specialOfferModule1 {
  height:286px;
  background:#ffffff url(../../data/reference/images/home/promo1-bg.jpg) no-repeat;
}

--------------------------------
-----------------
```

Note: The order in which the CSS files are included and classes are defined affects the application of styling rules. In case of multiple rules, the last rule will override the previous one. This is the basis on which these rules are applied.

CSS Folder Structure

The following figure displays the CSS file convention and directory structure:



Exploring a Theme

All the theme-related styling rules are kept in one copy of the `theme.css`. They are rules in terms of font, color, and background images that have a great impact on the look and feel of the display.

Theme related styling rules can be categorized according to the way in which they are organized and applied:

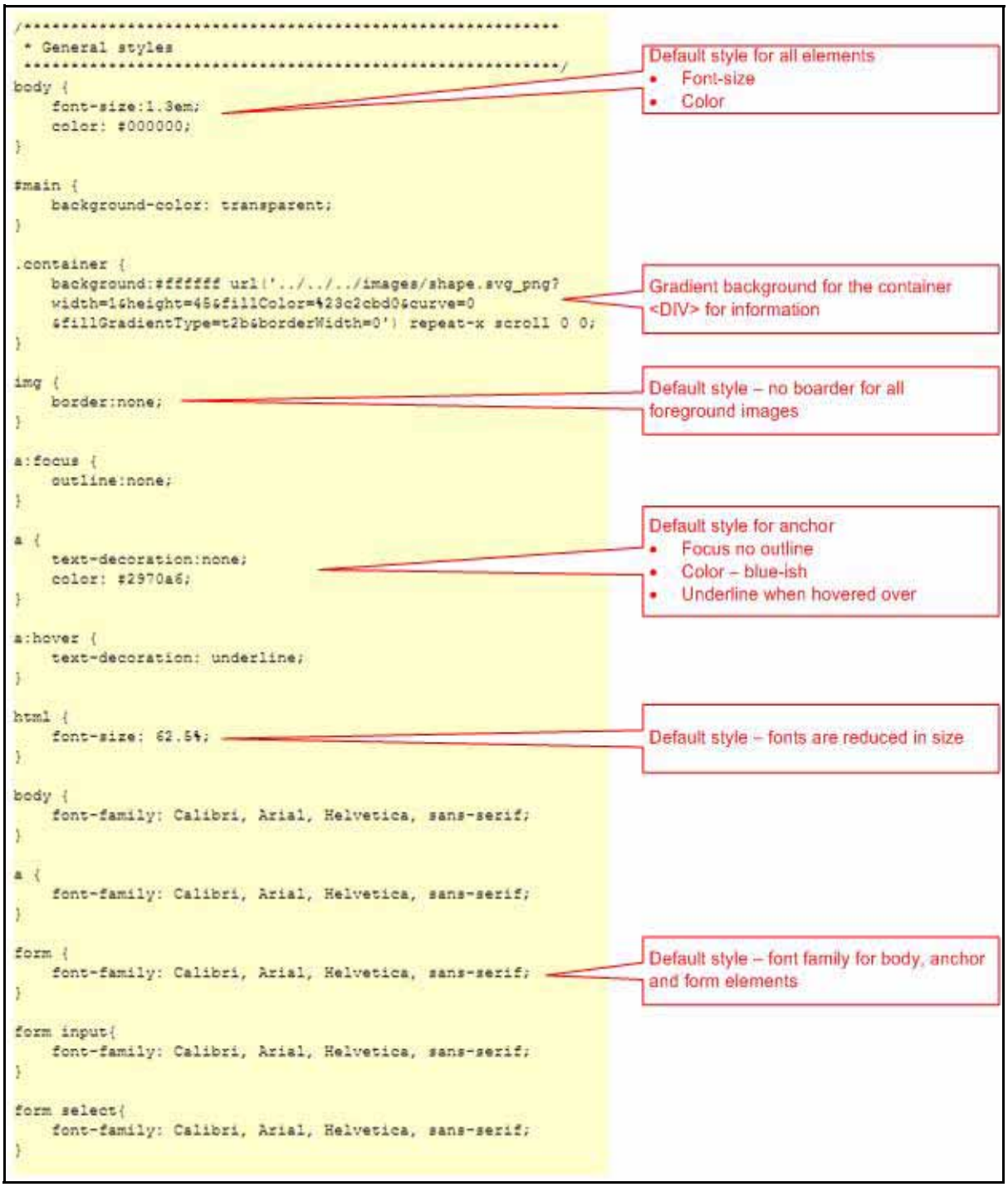
- Default style for HTML elements
- Style for common UI patterns
- Style for UI components
- Page-specific style

For more information on how to create a custom theme file, refer to the *Sterling Web Customization Guide*.

Default Styles for HTML Elements

There are rules used to define default styles for HTML elements, such as the anchor tag `<a>`, elements in a form and so on.

As illustrated in the following figure, the font family used for all form elements is defined as Calibri, Arial, Helvetica and Sans Serif. The font color for all anchor tags <a> is defined as #2970a6, which is a shade of blue. Unless overwritten, these styles cascade throughout all the page displays. This is how the basic display factors in the Sterling Web UI are enforced.



Typically, these rules should not be overwritten in individual page designs. Certain UI patterns or UI components might have their specific style requirements. For example, the font color for links in the pagination control should be white to provide better contrast. The designs and decisions of the changes in each theme must be carefully reviewed so that consistency of the look and feel can be easily maintained.

Styles for Common UI Patterns

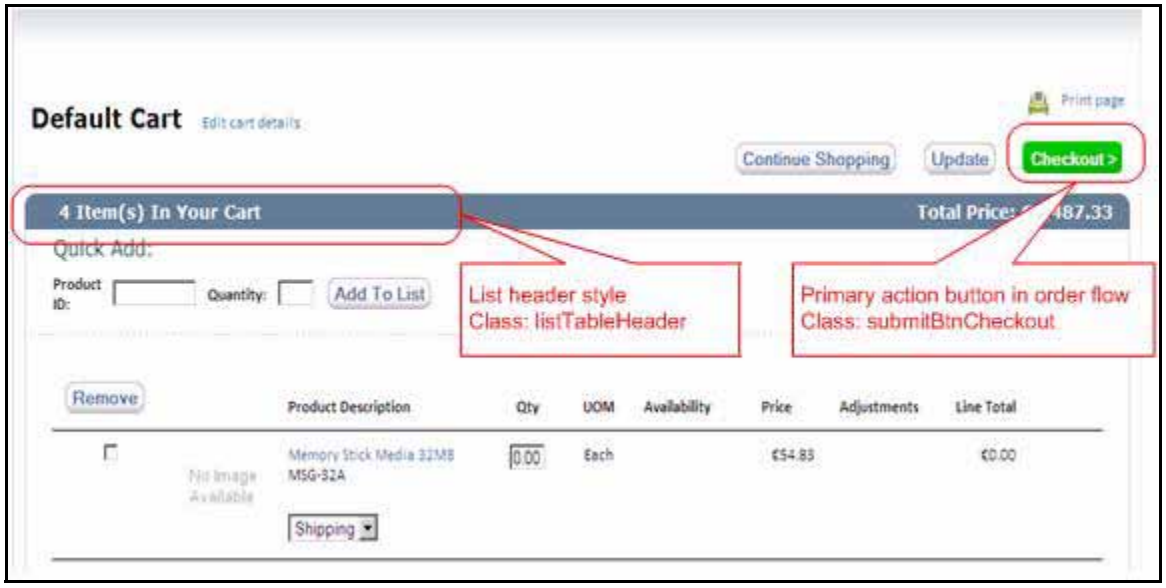
Common UI patterns are repeated all over the displays in the Sterling Web UI. For example, various types of command buttons, table headers with the same type of text, color, size, and background color and so on. These patters are typically implemented with the same HTML coding patterns in JSPs, with the proper styles assigned. This is another way of enforcing a consistent look and feel in the Sterling Web UI.

- Command buttons - One common UI pattern that is followed is various types of command buttons. In each page, there could be actions with different levels of significance that are triggered by clicking buttons with different visual effects.

  As illustrated in the following figures, buttons with different color, font size, and background color are used to differentiate actions.

```
.submitBtnBg1
{
  background: #bbbbbb;
  color: #5a70a6;
  font-weight: bold;
  cursor: pointer;
}

.submitBtnBg2
{
  background-color: #005ca3;
  color: #ffffff;
  font-weight: bold;
  cursor: pointer;
}

.submitBtnBg2:hover {
  background-color:#0d73c2;
}
```

```
.submitBtnCheckout
{
        background-color: #00cc00;
        color: #ffffff;
        font-weight: bold;
        cursor: pointer;
}

.submitBtnCheckout:hover {
        background-color:#0d73c2;
}
```

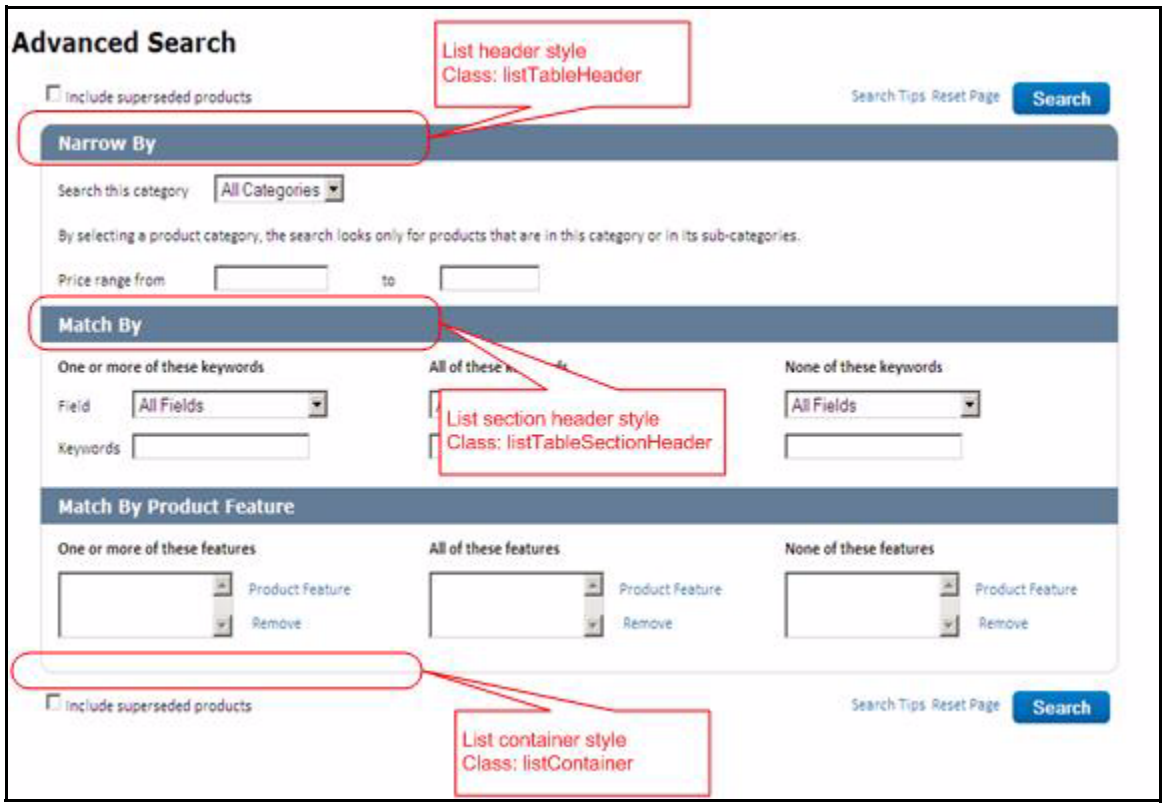


- List table - This is another type of UI pattern followed in Sterling Web. One example of the usage is illustrated in **Catalog Advanced Search**, as displayed in the following figures. The default font family, color and size for text, and the background color in the header and the divider are considered theme-related and captured in `theme.css`.

  The following figure illustrates the signs defined for the list tables.

```
/*****************************************************************************
Component: listTable
List table is the general UI component using table element to display a list
of entities with various type of detail inside table cells.
*****************************************************************************/

.listTableContainer {
  border:none;
}

.listTableHeader, .listTableHeader2
{
        background-color: #637d96;
}

.listTableSectionHeader
{
        background-color: #637d96;
}

/* Define the font style for the header text in the primary table header */
.listTableHeader .listTableHeaderText
{
        color:#ffffff;
        font-family: Verdana, Geneva, sans-serif;
        font-weight: bold;
        font-size: 1.1em;
    text-transform:capitalize;
}

.listTableSectionHeader .listTableSectionHeaderText
{
        color:#ffffff;
        font-family: Verdana, Geneva, sans-serif;
        font-weight: bold;
        font-size: 1.1em;
        text-transform:capitalize;
}

.listTableHeader2 td
{
        font-size: 1.0em;
}

/* Define the font style for the header text in the secondary table header */
.listTableHeader2 .listTableHeaderText
{
        color:#ffffff;
        font-family: Verdana, Geneva, sans-serif;
        font-weight: bold;
        font-size: 1.0em;
}

.borderRight
{
        border:1px solid #cccccc;
        border-style:outset;
        border-collapse:separate;
}
```

The following figure illustrates a list table used in **Catalog Advanced Search**:



Component Style

UI components are well-defined code-constructs that are invoked for their predefined behaviors and displays. These components are typically packaged into libraries such as the Struts library, the JSP tag library or the JavaScript library. They are included in JSPs with proper declarations and instantiations. Some of these libraries are built by Sterling Commerce, and they the same style of the engineering process. Third-party components are used too, for example, some UI components from Ext Js are used in the Sterling Web UI.

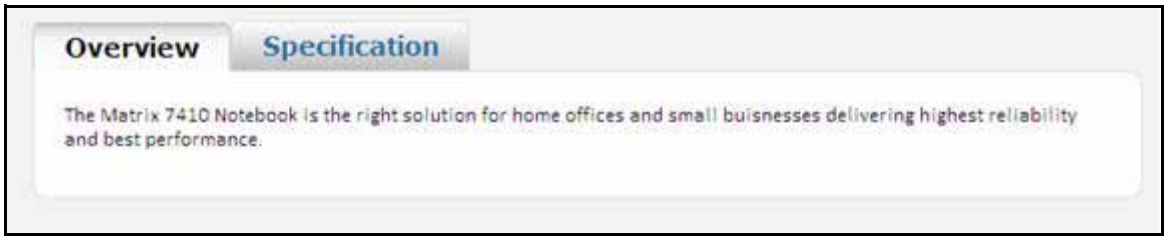
Sterling UI Components

- Tab control

  The tab UI used in Sterling Web is implemented in two ways:

  - At the client-side with JavaScript
  - Through JSP fragments refreshed by request

However, the styles for the tab UI are consistent throughout all the page displays, regardless of how the styles are implemented. The following figure illustrates the tab control used in the Catalog Detail page.

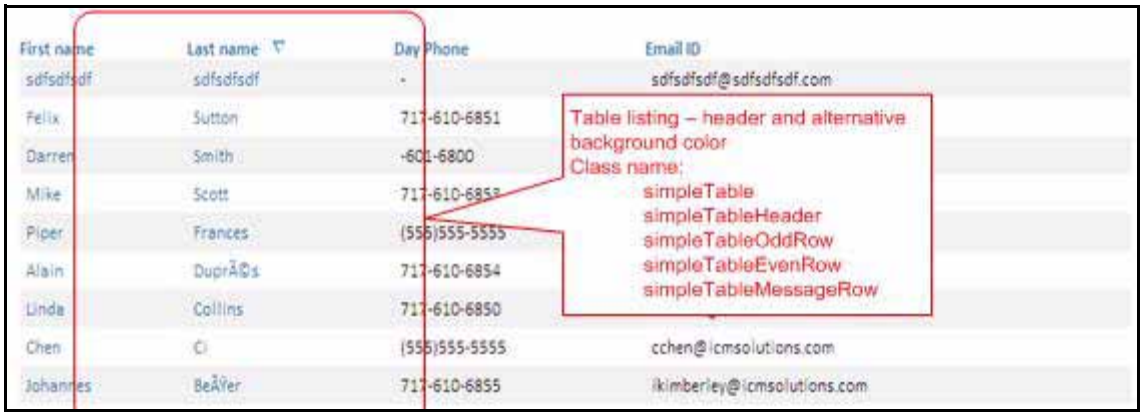


The following figure illustrates the theme-related styles for tab control.

```
/*****************************************************************
 Tab 1 Styling
*****************************************************************/

.tab-1 li {
        border-color: #d0d0d0;
        background-color: #ffffff;
}

.tab-1 a {
        text-decoration:none;
        text-transform:capitalize;
}

.tab-1 a:hover {
        /*background: #dfffdd;*/
}

.tab-1 .selected {
        border-color: #d0d0d0;
  background: #ffffff;
}

.tab-1 .selected a {
        color: #000000;
        font-weight: bold;
}

.tabContent {
        border: 1px solid #f2f2f2;
        background-color: #ffffff;
}

.tab-1 a
{
        font-family: Verdana, Geneva, sans-serif;
        font-size: 1.0em;
        font-weight: bold;
}
```

- Simple table

  Another example of UI component is the simple table in the Sterling Web UI. This component is implemented as an extension to the Struts tag by Sterling Commerce. Although invoked as a custom tag, the styles controlling the look and feel are captured as CSS and put into `theme.css`, as illustrated in the following figure:



The following figure illustrates the styles used by the simple table control.

```
/*******************************************************************************
 * Simple Table Component Styling                                              *
 *******************************************************************************/
.simpleTable {
}
.simpleTableHeader {
        font-weight: bold;
}
.simpleTableOddRow {
        background-color: #f2f2f2;
}
.simpleTableEvenRow {
        background-color: #ffffff;
}

.simpleTableMessageRow {
        text-align: center;
}
```

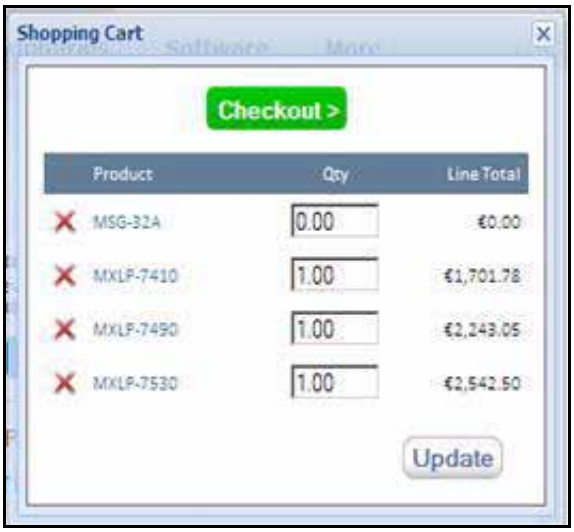
Third-Party components

Sterling Web uses UI components built by third parties. Sterling Commerce subscribes to the use of Ext Js as the main JavaScript library for its utility functions and UI components. When compared to other Sterling application suites, Sterling Web UI uses relatively less client-side UI components, but it does adopt a few, such as Ext.Window, Lightbox, and so on.

Using CSS to capture look and feel is the mainstream technique employed in the Web UI world. Typically, UI components from vendors come with associated styles, and are usually organized into CSS files. To enable these UI components blend in, these styles must be customized to adopt the same color scheme, font styles, and so on as the other UI elements in Sterling Web. One technique to achieve that is to replicate all the style rules or classes in the vendor CSS files to `theme.css`, and provide the appropriate values. By including only the `theme.css` in a JSP, or including the `theme.css` after the vendor CSS, the vendor styles will be overwritten, and the desired look and feel reflected.

- Ext Window

  The following figure illustrates how an Ext.Window from Ext Js used to display a shopping cart:

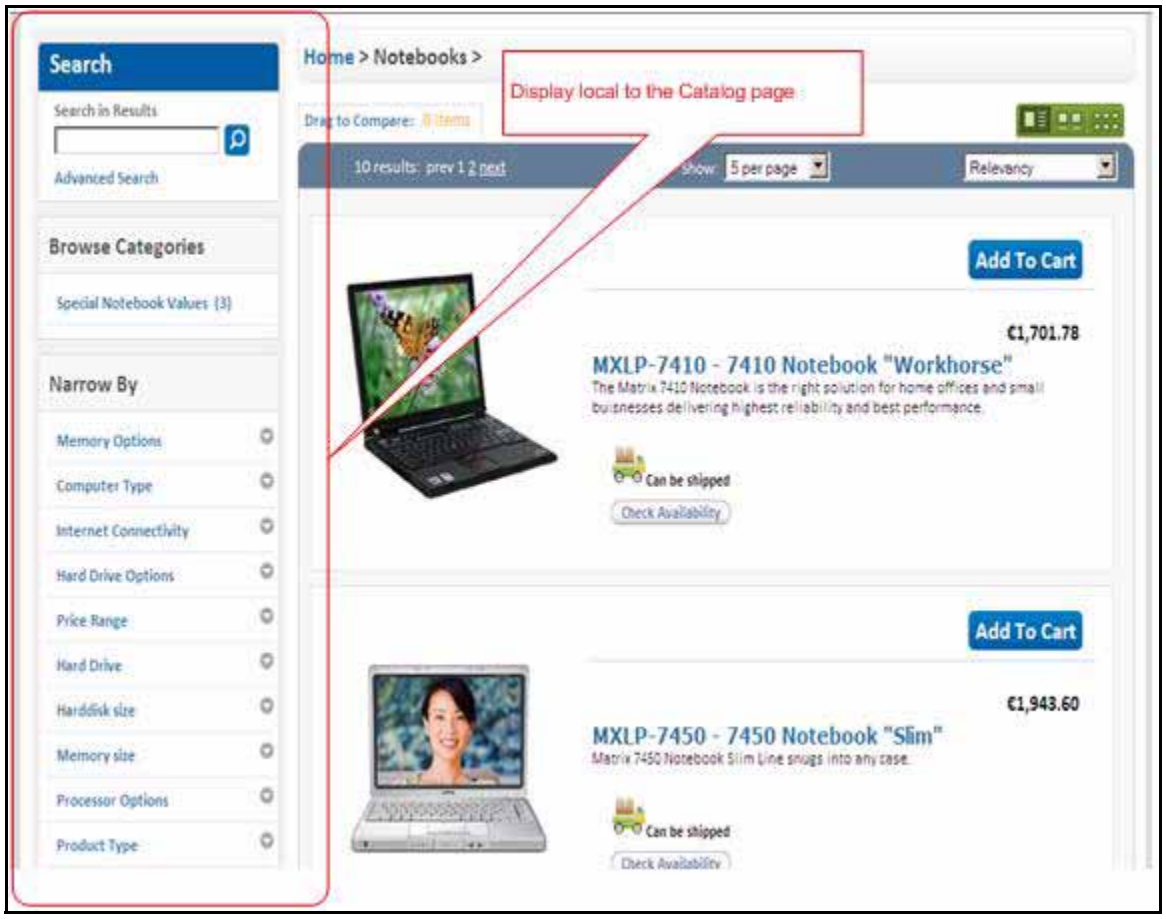


The following figure illustrates the styles used by Ext.Window that can be overwritten in a theme:

```
/*****************************************************************************
 * Styles used to overwrite Ext window
 * Usage: set property "cls" with value "swc-ext" when create the Ext.Window
 *****************************************************************************/
.swc-ext .x-window-tl .x-window-header{
    color:#15428b;
    font-weight: bold;
    font-size: 1.0em;
    font-family: Calibri,arial,verdana,sans-serif;
    padding:5px 0 4px 0;
}
.swc-ext .x-window-tc{
  background:transparent url(../../../ext/resources/images/default/window/top-bottom.png) repeat-x 0 0;
  overflow:hidden;
}
.swc-ext .x-window-tl {
  background: transparent url( ../../../ext/resources/images/default/window/left-corners.png ) no-repeat 0 0;
  padding-left: 6px;
  z-index: 1;
  position: relative;
}
.swc-ext .x-window-tr {
  background: transparent url( ../../../ext/resources/images/default/window/right-corners.png ) no-repeat right 0;
  padding-right: 6px;
}
.swc-ext .x-window-bc {
  background: transparent url( ../../../ext/resources/images/default/window/top-bottom.png ) repeat-x 0 bottom;
}
.swc-ext .x-window-bc .x-window-footer {
  padding-bottom: 6px;
  font-size: 0;
  line-height: 0;
}
.swc-ext .x-window-bl {
  background: transparent url( ../../../ext/resources/images/default/window/left-corners.png ) no-repeat 0 bottom;
  padding-left: 6px;
}
.swc-ext .x-window-br {
  background: transparent url( ../../../ext/resources/images/default/window/right-corners.png ) no-repeat right bottom;
  padding-right: 6px;
}
.swc-ext .x-window-mc {
  border: 1px solid #99bbe8;
  padding: 0;
  margin: 0;
  font: normal 1.1em Calibri, arial, helvetica, sans-serif;
  background: #dfe8f6;
}

.swc-ext .x-window-ml {
  background: transparent url( ../../../ext/resources/images/default/window/left-right.png ) repeat-y 0 0;
  padding-left: 6px;
}
.swc-ext .x-window-mr {
  background: transparent url( ../../../ext/resources/images/default/window/left-right.png ) repeat-y right 0;
  padding-right: 6px;
}
```

Page-Specific Styles

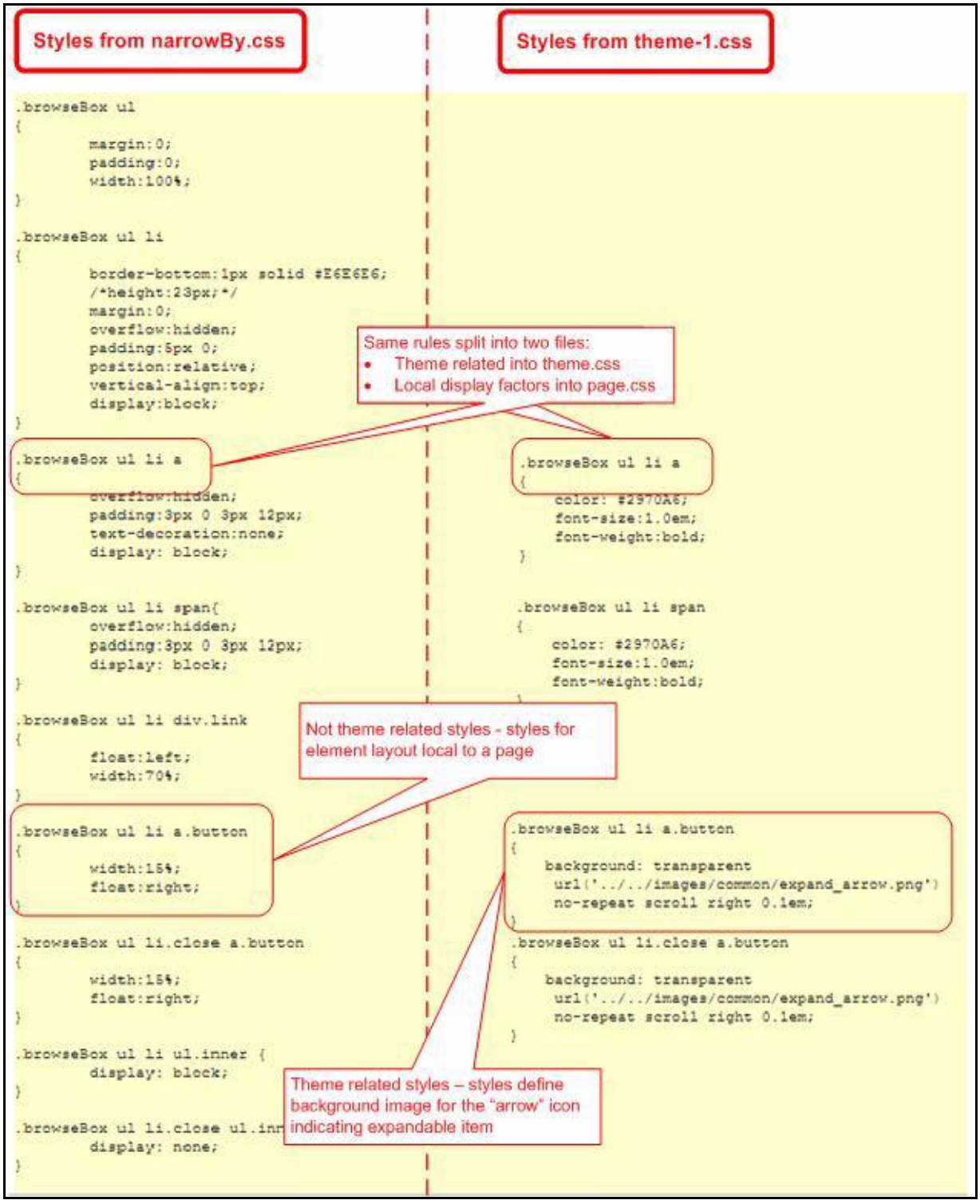
Specific styles are used in certain displays. Because they are significant to the look and feel of the page displayed, they are captured in `theme.css` as well. One such example is the Search, Browse Categories, and Narrow By widgets displayed in the left column of the Catalog page, as illustrated in the following figure:



In any given page, most of the theme-related styles are already defined in `theme.css` in terms of the styles described in the previous sections- styles for HTML element, styles for common UI patterns, and UI components. There are other theme-related styles that are not captured in the categories mentioned during page design. They must be put into `theme.css` so that they can be controlled.

One common technique is to split the styles that are used in page design into two parts based on whether they are theme-related or not. Classes with the same name exist in `theme.css` to define the theme-related aspect of the styles and in page-level CSS files to define the rest of the required styles.

The following figure illustrates how some parts of the classes are promoted into `theme.css`.



Performing Common Theme-Related Modifications

This topic provides information about performing a few common UI-related tasks. These tasks can be segregated to two categories based on what they change:

- Related to themes
- Not related to themes

Theme-related UI changes can be loaded based on Web stores, whereas changes not related to themes are usually reflected on displays across all Web stores.

The tasks pertaining to modifying theme-related UI are:

- Changing or Creating Styles for a Web Store
- Changing Theme-Related Styles
- Changing the SVG-Related Visual Effects

Tasks that are non-theme-related include the following:

- Tweaking the Template Size
- Adding New CSS Class
- Using New UI Components

Examples of where and what to change for each task are given in the context of the engineering source tree. The field team should follow the same procedures described in this document but keep the artifact of any customization in accordance with process detailed in the *Customization Guide*.

Changing or Creating Styles for a Web Store

Sterling Web UI has the built-in capability of serving each Web store with a different look and feel. This is achieved by loading Web store-specific `theme.css` based on the assignment of the `theme.css` to each Web store by the administrator through Sterling Business Center.

A particular theme can be modified by changing the corresponding `theme.css`. The result will be reflected in all the Web stores adopting that theme. When a new Web store is created, the administrator has the options of using a set of existing styles for the new store, or using a set of newly created styles. The discussion in this section focuses on how to create a set of new styles for a new Web store. Refer to the *Customization Guide* and *Administration Guide* for information about how to create new Web stores and manage the assignment of themes.

To create a new theme for a Web store:

1. Make a copy of `theme-x.css` from the existing ones and save it in the store-specific directory.
2. Assign `theme-x.css` to a Web store X in Sterling Business Center.
3. Change the styles in `theme-x.css` and test the visual impact.

Changing Theme-Related Styles

Styles pertaining to fonts, colors, and background images are controlled through `theme.css`. One can modify the look and feel of a Web store by changing the related theme defined in `theme.css`.

For example, following is the procedure to change the font family, size, and color of all the anchor links:

1. Identify the target copy of `theme.css`.
2. Change the following lines:

```
a
{
text-decoration:none;
color: #094B53;
}
a
{
font-family: Calibri, Arial, Helvetica, sans-serif;
}
```

For example, following is the procedure to change the background color of a list table's header:

1. Identify the target copy of `theme.css`.
2. Change the following highlighted lines:

```
.listTableHeader, .listTableHeader2
{
  background-color: #637D96;
}
.listTableSectionHeader
{
  background-color: #637D96;
}
```

Note: `listTableSectionHeader` defines the styles for the section divider in the list tables. It probably makes sense to change the color together with the `listTableHeader`.

For example, following is the procedure to change the background images for the anchor links used in the UI component vertical filter. This is done by selecting **NarrowBy** in the dropdown menu.

1. Identify the target copy of `theme.css`.
2. Change the following highlighted lines:

```
.vertical-filter ul li a.button {
  width:15%;
  float:right;
  background: url(../../images/theme/theme-1/vertical-filter/arrow-3.gif)
no-repeat 15px -71px;
}
.vertical-filter ul li.close a.button {
  width:15%;
  float:right;
  background: url(../../images/theme/theme-1/vertical-filter/arrow-3.gif)
no-repeat 15px 12px;
}
```

Changing the SVG-Related Visual Effects

The Sterling Web UI utilizes JavaScript in combination with back-end service images rendered with Scalable Vector Graphics (SVG) technology to decorate Web pages dynamically. The technique is used to provide visual effects such as background images with gradient for certain areas in a Web page or for an entire Web page, rounded corners for some UI elements, and background images for some UI components.

For example, following is the procedure for creating new button styles:

1. Assign a class name, say btn1 for this new button style.
2. Identify the `theme.css` that is specific to the target Web store or theme.
3. Search for the `decorateButtons()` function.

4. In the function, add a new rule by copying it from an existing rule.

```
rules+=SVGAnnotator.mkGeneralShapeRuleSet(".submitBtnBg1", {
            cssClass:'submitBtnBg1BG',
            width:null,
            height:null,
            fillColor:"#bbbbbb",
            fillGradientType:"t2b",
            outsideColor:null,
            outsideOpacityPct:100,
            shadingStyle:'sine',
            shadingLevelPct: 98,
            borderWidth: 1,
            borderColor: '#bbbbbb',
            curve: 8}, ":hover");
```

5. Change class name `.submitBtnBg1` to `.btn1`
6. Make other changes, as required.
7. Assign the `btn1` class to all the HTML elements that will display this button style.

Following is the procedure for creating rounded corners for an area defined by a `<DIV>`:

1. Assign a class name, namely area1, to the `<DIV>`, if not existing.
2. Identify the `theme.js` that is specific to the target Web store (or theme).
3. Search for the `decorateBoxes()`function.
4. In the function, select a rule that meets your requirements, for example

```
rules += SVGAnnotator.mkGeneralShapeRuleSet('.subPanelBox, .groupPanelBox
.content', {
            cssClass:'subPanelBoxBG',
            width:null,
            height:null,
            fillColor:null,
            outsideColor:null,
            outsideOpacityPct:100,
            borderWidth: 1,
            borderColor: null,
            curve:5
        });
```

5. Insert the class name `.area1` into the first argument of the `mkGeneralShapeRuleSet` function.
6. You can also create a new rule by copying from the existing ones and making the necessary changes.

Note: If the JavaScript in pages resize panels with SVG backgrounds, the JavaScript functions should invoke `svg_classhandlers_decorate_page()` to regenerate correctly scaled backgrounds.

Tweaking the Template Size

Template-level CSS control the layout of page displays in the Sterling Web UI. Because template-level styles are shared, changes to these styles should be made after careful planning and design.

For example, to move the vertical navigation menu from the left side of the page to the right side, increase the size by 10 percentage.

1. Display impacted Catalog and user home pages.
2. Identify template to tweak namely `~/css/template/template-1.css`.
3. Analyze and verify the dependency. Search `~/css` and `~/jsp` for any inclusion. The following CSS files are obtained as the search result:
   - `~/css/catalog/catalogExt.css`
   - `~/css/home/portalhome.css`
4. Change the following:

```
.t3-main-content
{
float:right; /* changed from
float: left; */
  width:565px;
  padding:0;
  margin:0;
}
.t3-vertical-filter
{
  float:left;/* changed from
float: right; */
  margin-left: 10px;
  width:200px;
  padding:0;
}
```

Adding New CSS Class

When new styles are required, it is best to create them by following the conventions and processes. It benefits continuous development and maintenance.

Following are the different levels of abstractions for style rules:

- Template
- Theme
- Common
- Page levels

Note: When adding a new CSS class, it is essential to follow the same abstractions and place the different aspects of style in the correct places.

To add a new CSS class to a banner:

1. Create the CSS class in a page-level CSS file and test the visual impact.

   For example, this is the content of `page1.css`:

   ```
   .banner {
     width: 250px;
     height: 70px;
     padding: 5px 5px 5px 5px;
     float:right;
     background: url(../../images/theme/theme-1/banner1.gif) no-repeat 0 0;
   }
   ```

2. Verify whether there are template-level layout rules.
3. Move the theme-related attribute to `theme.css`.

   ```
   .banner {
     background: url(../../images/theme/theme-1/banner1.gif) no-repeat 0 0;
   }
   ```

4. Make sure the banner image is properly placed under the theme specific directory
5. As the result, what remain in `page1.css` are

   ```
   .banner {
     width: 250px;
     height: 70px;
     padding: 5px 5px 5px 5px;
     float:right;
   }
   ```

6. Replicate the setting to all the other `theme.css`, including the banner image, to each theme-specific directory.
7. Make theme-specific changes to each theme, if required.

Using New UI Components

One of the challenges of adopting a prebuilt UI component is to ensure that it blends with the corresponding Web pages. The customization process typically involves changing the default styles that come with the components. Other than changing the visual effect to what is expected, ensure the CSS styles are partitioned correctly to follow the abstractions and conventions. The main goal here is to make some of the desired visual effects themeable.

Revisit the section “Third-Party components” as an example.

Following is the procedure:

1. Include the default CSS files pertaining to the component into the target JSP as instructed. Test the result.
2. Examine the default CSS files and identify the styles necessary to be make them themeable.
3. Copy those styles into a copy of `theme.css`.
4. Make necessary changes to these styles to create the correct visual effect.
5. Ensure that `theme.css` is included after the default CSS file for the components in the target JSP. This will allow the styles in `theme.css` to overwrite the same styles defined in the default CSS.
6. Replicate the definition to all other `theme.css`.
7. Make theme-specific changes to each theme, if required.

Index

C

H

P

U