



Summer Cart 4.0
Design Customization Guide

Introduction

Summer Cart can easily be customized to match your desired look and feel and this guide will show you how to do this right. First, we will cover the basics and explain you how layouts, boxes and skins work, and show you the options you have to customize a Summer Cart store through the Admin Panel. It is quite powerful, but if you seek to create something truly unique, you will want to create a skin. We will cover everything you need to know about using and extending existing skins, as well as making your own. We also explain our code style and the conventions we use, and advise you to follow them so your code stays in shape – and that will be important when you have to maintain and update your skins. Finally, our How Do I section provides quick answers to common questions.

Prerequisites

Before you begin, make sure you have the following prerequisites in place:

- A working Summer Cart installation;
- Access to the store's Admin Panel;
- Access to the store files (e.g. through FTP);
- Certain knowledge in HTML and CSS.

Contents

Introduction	2
Prerequisites	2
Contents.....	3
The Basics.....	4
Layouts.....	4
Boxes.....	8
Rich Text Pages	9
Dynamic Pages.....	9
Forms	9
Plugins.....	9
Menus	10
Skins	10
Working with Skins	11
Creating a New Skin	12
Skin Definition File	12
Skin Settings.....	13
Skin Settings in Multiple Languages	16
Skin Inheritance	16
Folder Structure.....	17
Template Structure	19
HTML and CSS Structure	22
Code Style and Conventions	27
How Do I.....	28
Customize the CSS of a site through the Admin Panel?	28
Set CSS rules for a specific layout?	28

The Basics

Layouts

Each page in Summer Cart follows the structure illustrated in fig. 1.1. Pages have a common header and footer that are the same across the website and are defined in a skin. Then, there are 4 areas between the header and the footer – the top, left, right, and bottom areas. These can be different for each page and are defined in a layout. Multiple pages can share the same layout. Finally, the actual content of the page (e.g. a grid of products) is displayed in the content area in the center.

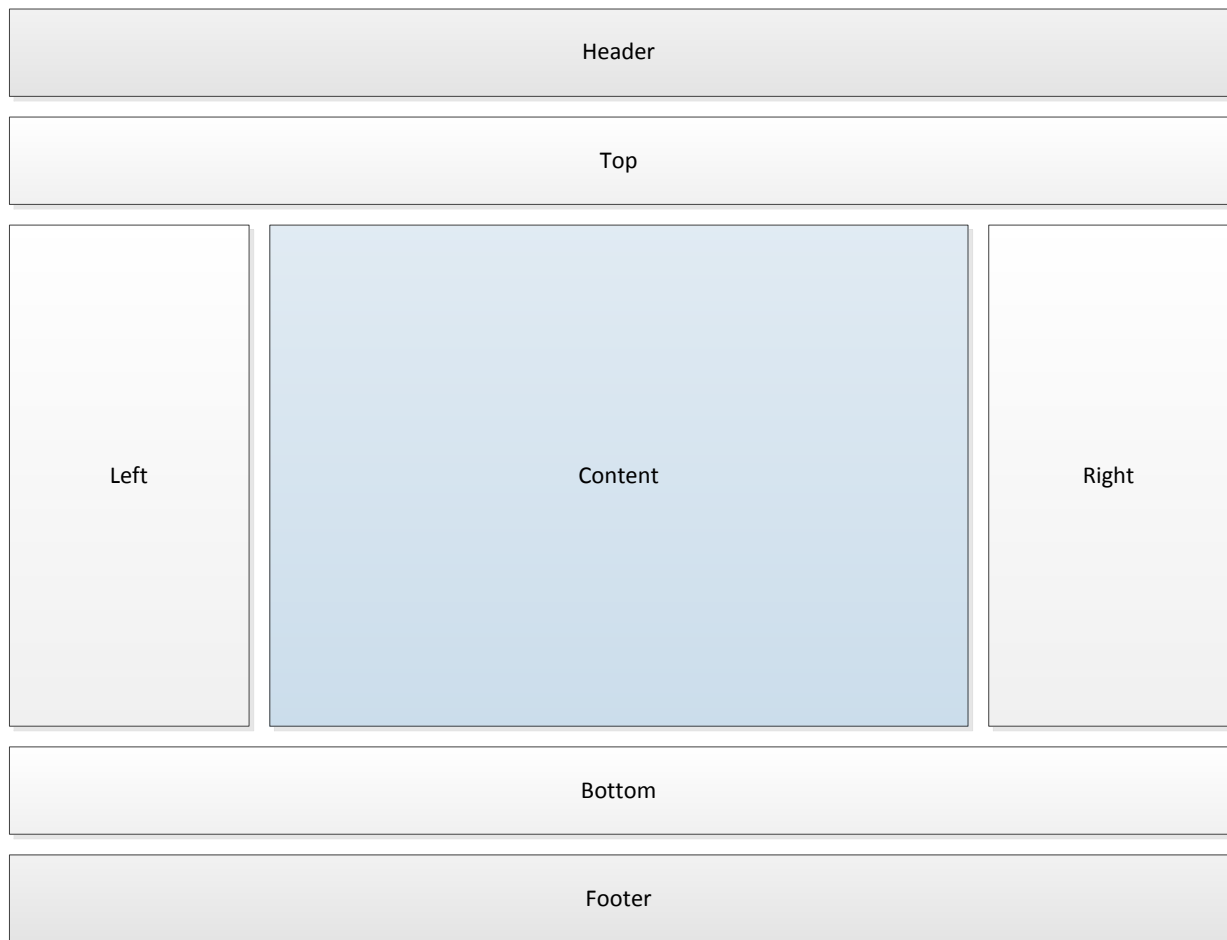


Fig. 1.1 Summer Cart page structure

Layouts are great for organizing your site’s elements into logical units (“boxes”) and reusing them across pages, sometimes in different order or in different formations. Through the Admin Panel you can organize boxes into a layout by putting them into the top, left, right, and bottom areas of the layout. There are a number of built-in boxes, custom HTML boxes that you create, and box menus. Boxes that are not active will not be visible in the layout designer.

Below is the default layout for our “Silverview” demo site. It is used by the Home page, and you can see a screenshot of it on the next page. Note the four areas of the layout being displayed as containers and how in the left and right area we have arranged a number of boxes. (This particular layout does not have any boxes in the top or bottom areas.)

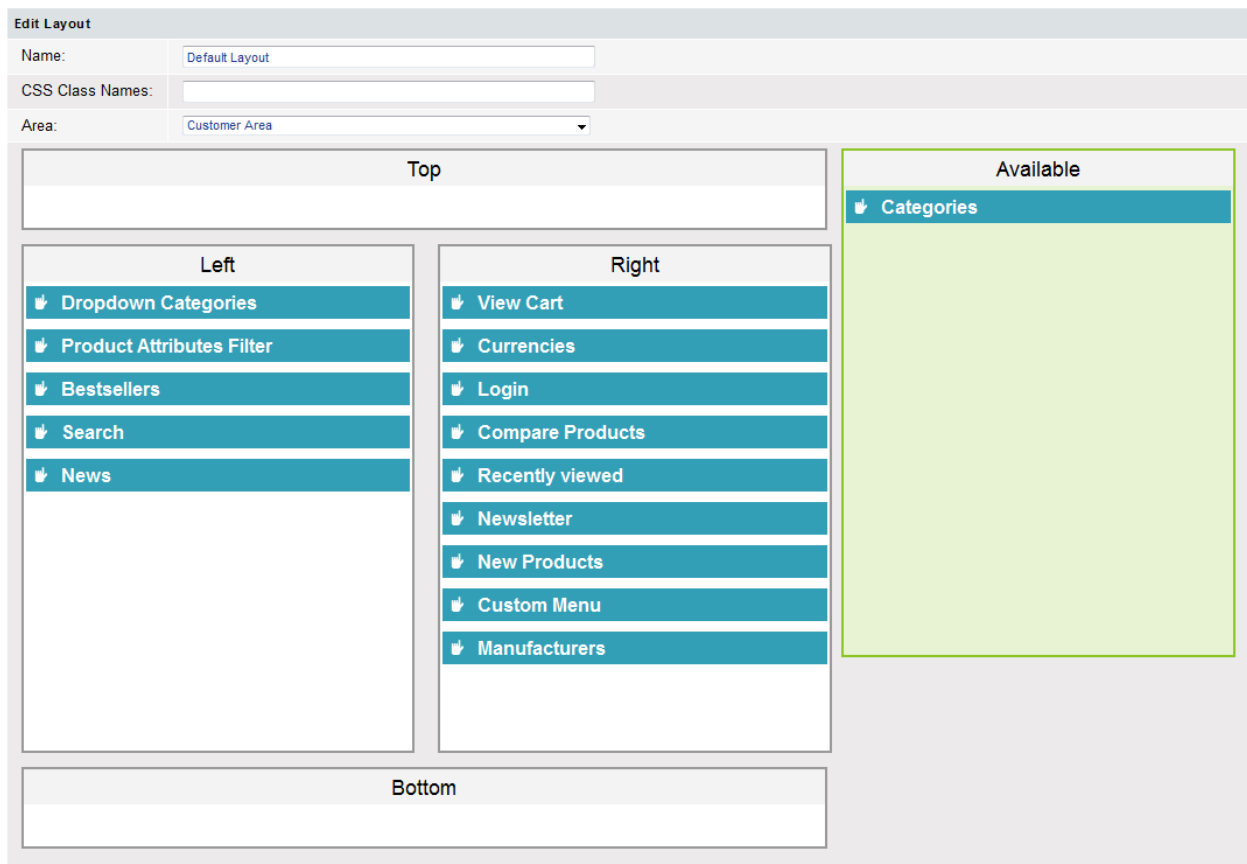


Fig. 1.2 A layout definition

Now compare the layout definition above with the actual page below. If you can’t spot at once the matching structure, on the next page you can see the structure highlighted.

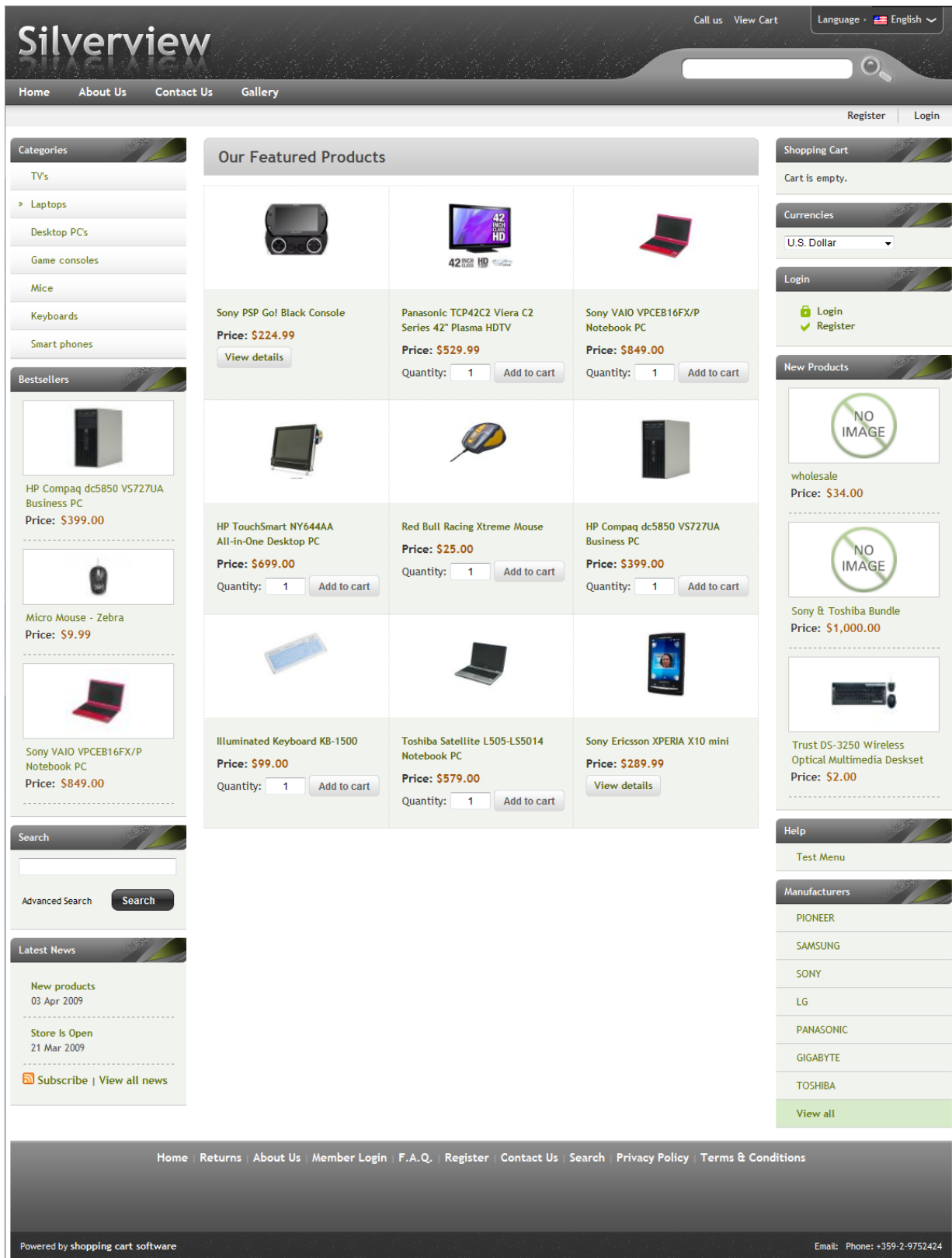


Fig. 1.3 Silverview Home page using the layout from the previous page

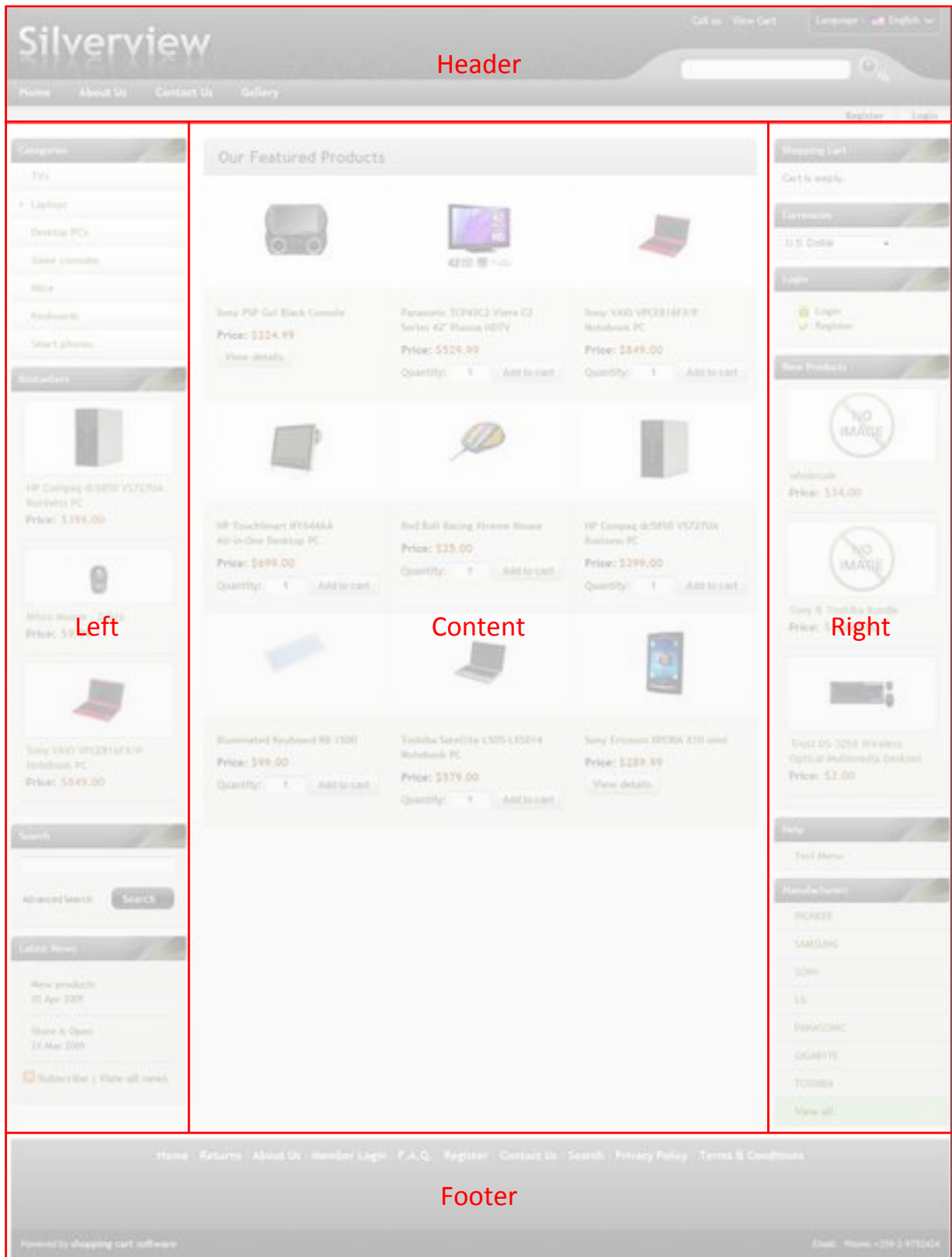
































Fig. 1.4 The same page with its layout highlighted for illustration

An important layout setting is the “CSS Class Names” field. Here you can type a comma-separated list of CSS classes and they will be added to the CSS classes of the body element of pages that use this layout. This would be useful if you have a custom skin, because it would allow you to write CSS rules that targets elements only if the page uses a specific layout.

Boxes

From the Boxes page in the Admin Panel you will see a list of all the boxes that are currently available in the system. Most of them will have one or more options you can customize. For example, the Bestsellers box has the options of what number of bestsellers to show, and what period of time (e.g. last 30 days) to use when calculating total sales for a product. You can also create your own custom HTML boxes and use them in layouts. Note that each box has its own ID, and if you create a custom skin you can set CSS rules for that specific box using its ID (more on that in the chapter “Working with Skins”).

New Box

Area	Name	Active	Type	
Customer Area	Bestsellers	✓	System Box	 
Customer Area	Login	✓	System Box	 
Customer Area	Categories	✓	System Box	 
Customer Area	Manufacturers	✓	System Box	 
Customer Area	View Cart	✓	System Box	 
Customer Area	Search	✓	System Box	 
Customer Area	News	✓	System Box	 
Customer Area	Currencies	✓	System Box	 
Customer Area	Recently viewed	✓	System Box	 
Admin Area	Setup Wizard	✓	System Box	 
Customer Area	New Products	✓	System Box	 
Customer Area	Dropdown Categories	✓	System Box	 
Customer Area	Product Attributes Filter	✓	System Box	 
Customer Area	Compare Products	✓	System Box	 
Customer Area	Newsletter	✓	Newsletter Box	 

Boxes 1 - 15 of 15 First < Previous < 1 > Next > Last

Fig. 1.5 List of built-in boxes

For maximum flexibility with your custom HTML boxes, you can uncheck the “Show Title” checkbox, in which case the only thing Summer Cart will add to your custom HTML is put it into a div with the ID of the box, so you can address your box in your CSS rules.

At any time you can easily remove a box from all layouts that include it by unchecking the box’s Active checkbox. This way the box will disappear from all layouts and the Customer Area, but as soon as you activate the box again it will reappear in its original place.

Rich Text Pages

Rich Text Pages are static HTML pages you can add through the Admin Panel. As with all pages, these can have different layouts. To apply CSS you will have to either embed the CSS to the page, or create a skin.

Dynamic Pages

Dynamic pages are ones built into the system, with their content generated dynamically. These are for example the Home page, the Products page, News page and so on. Dynamic pages use HTML templates, and PHP scripts parse these templates and generate the actual HTML code. From the Admin Panel you can add static HTML and CSS code to their tops and bottoms, and change their layout. However, to fully customize their HTML or CSS (not just add code to tops and bottoms) you will have to create a skin.

Forms

You use forms when you want to collect feedback from the customers. The most ubiquitous example is a Contact Us form. A form has a title and description, and a number of input controls such as your regular textboxes, multiline textboxes, checkboxes, radio buttons and drop-downs. You add these through the Admin Panel, without having to write any HTML code. In addition to these, you can use a number of built-in input controls with specific function, such as email and number textboxes, countries and states drop-downs, file upload control and a custom HTML control. The input from the user is sent through email to the email address set in the form settings.

Plugins

Plugins are packages of extra features that can be added to a Summer Cart store when the user installs a plugin through the Admin Panel. A plugin may contain additional templates, and overwrite existing templates. A plugin may also contain additional modules that have their own templates and CSS files.

Menus

In Summer Cart you are not restricted on the number of menus you can have on a site. Typically you would have a Main Menu displayed in the header of the site, with links to pages like Home, About Us, Contact Us, etc. Another place often used for a menu is the footer of the site with links to F.A.Q. page, Privacy Policy, Terms & Conditions, and others. In Summer Cart there are 4 possible menu locations:

Menu Location	Description
Top	Menu is displayed in the site header.
Footer	Menu is displayed in the site footer.
Fast Menu	Menu is displayed in the site header, typically on the very top of the site, separate from the Top menu.
Box	Menu is displayed in a box. When you add a new menu with Box location, a new box will be added to layout elements, one with the name of your new menu.

In a menu you can add a number of menu items. Each menu item is a link to either a Rich Text Page, Dynamic Page, URL, Form, or a Gallery. Menu items can also have icons that you upload.

Skins

Skins contain HTML templates for pages and boxes, CSS files and images, and script files. At any time you can change the look and feel of a Summer Cart store by changing its skin. Note that this will not affect layouts, which work independently from skins. For example, if you put a Search box in the Top area of a layout, even if you change the skin the Search box will remain in the Top area, although it may look completely different.

From the Admin Panel you can see a list of all the skins installed on the site. Some of the skins have settings that you can customize by clicking the Settings link next to the skin's name. Skins are covered in great detail in the next chapter.

Working with Skins

Summer Cart ships with four built-in skins for the Customer Area, and one for the Admin Area. You can change the current skin of the site from Admin Panel > Website Content > Skins.

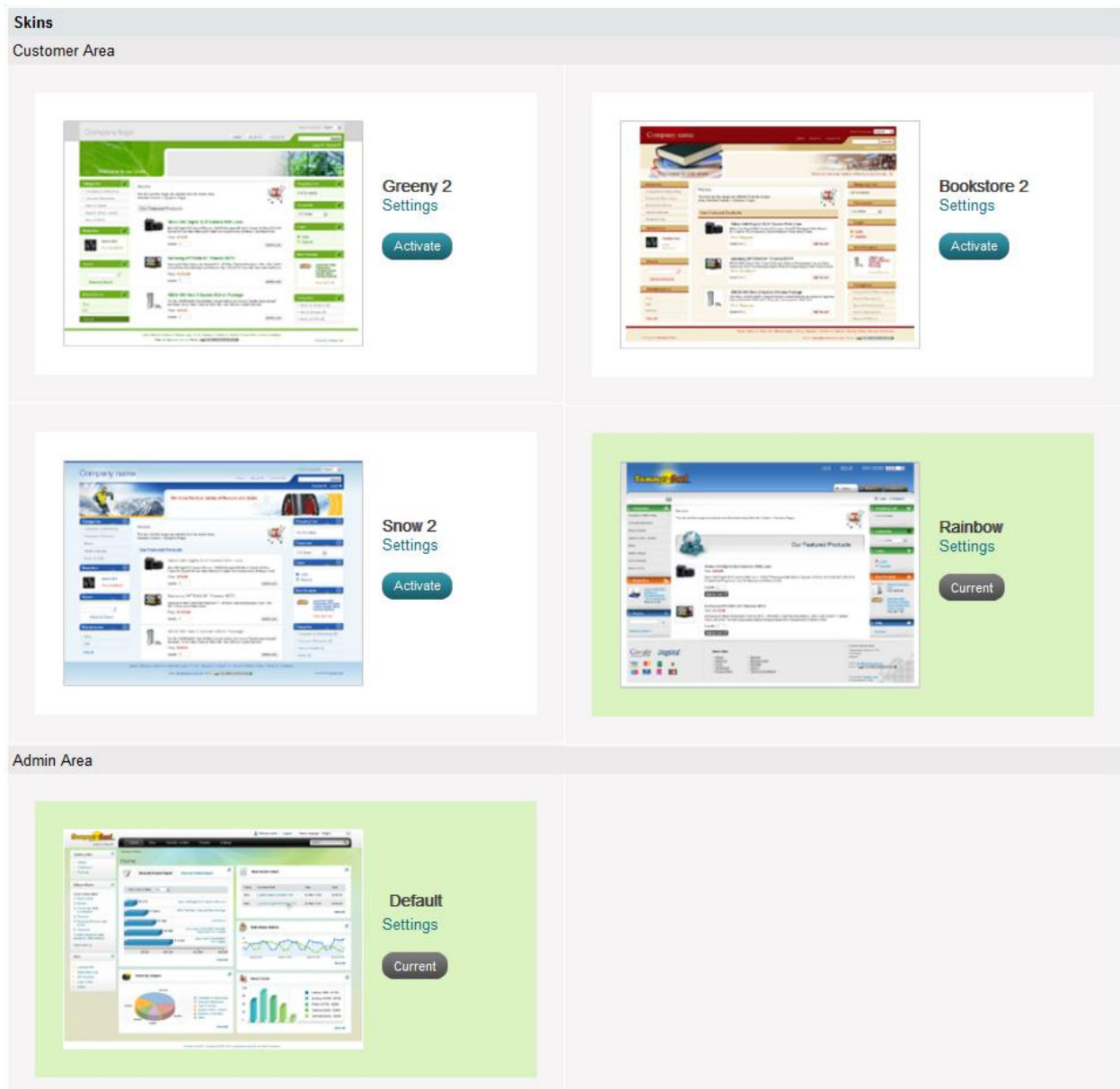


Fig. 2.1 List of installed skins

If you look at a Summer Cart store through an FTP browser, you will see there is a /skins folder. This is where skins are located. By default, it has 6 subfolders:

- /skeleton** – this is the basic skin. All Customer Area skins inherit from the skeleton skin;
- /default** – the default skin for the Admin Panel. Normally you shouldn't modify it;
- /bookstore2** – the Bookstore 2 skin;
- /greeny2** – the Greeny 2 skin;
- /rainbow** – the Raindow skin;
- /snow2** – the Snow 2 skin.

Creating a New Skin

To create a new skin, simply create a new folder in /skins, for example "myskin". You are strongly advised to create a new skin by copying one of the existing 4 skins. This will save you a lot of work as these skins contain much CSS and HTML that you can simply customize, and not start from scratch.

Skin Definition File

All skins must have a skin definition file named skin.xml. Create one for your new skin as well. Let's look at the contents of this file (I used the Greeny 2 skin for illustration):

```
<?xml version="1.0" encoding="UTF-8"?>
<Skin>
  <SCVersion compare="equal">4.0</SCVersion>
  <Name>Greeny 2</Name>
  <SkinLocation>greeny2</SkinLocation>
  <Screenshot>skin-screenshot.png</Screenshot>
  <Areas>
    <Admin>>false</Admin>
    <Customer>>true</Customer>
  </Areas>
  <SkinParentLocation>skeleton</SkinParentLocation>
  <Settings>
    <Setting type="image" name="Logo" displayName="Logo" maxWidth="400"
maxHeight="200">customer/images/logo.png</Setting>
  </Settings>
</Skin>
```

Property	Description
SCVersion	The version of Summer Cart this skin is compatible with. Must match the currently installed version.
Name	Name of the skin.
SkinLocation	Location of the skin inside the /skins folder.
Screenshot	Screenshot of the skin, used on the Skins page in Admin Panel.
Areas	Whether the skin is for the Customer Area, the Admin Panel or both. Typically you would want your skin to be for the Customer Area only.
SkinParentLocation	The parent skin. This would typically be the skeleton skin.
Settings	Customizable skin settings that can be modified through the Admin Panel (see below).

Skin Settings

If you want your skins to be customizable through the Admin Panel, you must use Skin Settings. These are settings you define in your skin.xml file, and users can set their values from the Admin Panel. Then you can use these values in your templates. Let's look at the settings of the Greeny 2 skin from the previous page for illustration:

```
<Settings>
  <Setting type="image" name="Logo" displayName="Logo" maxWidth="400"
maxHeight="200">customer/images/logo.png</Setting>
</Settings>
```

This defines a setting named Logo, of type image (one that the administrator can upload), and specifies that the uploaded image must have max size of 400x200 pixels and its default value would be /customer/images/logo.png. Now if we go to the Skins page in the Admin Panel and click on Greeny 2's Settings link, we will see our setting:

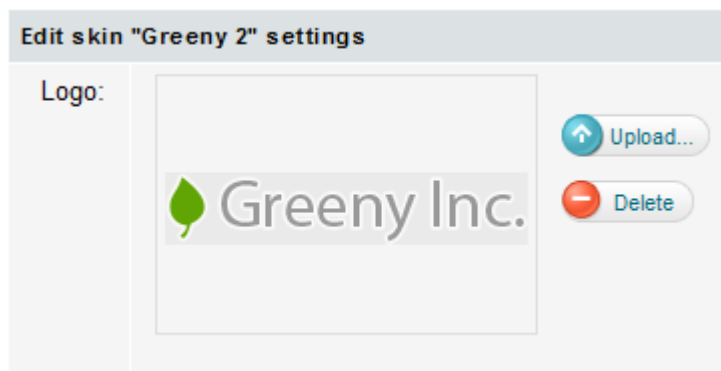


Fig. 2.2 “Greeny 2” skin settings

Note that if the administrator deletes the value (clicks the Delete button next to the picture), the default value for the setting (in this case /customer/images/logo.png) will be used.

There are 3 types of settings supported by Summer Cart 4.0. These are:

Setting Type	Description
text	A text setting. Actual value would be the text entered by the user.
checkbox	A Boolean setting. Actual value would be either 1 (true) or 0 (false) depending whether the user checked the checkbox.
image	An image setting. Actual value would be the relative path to an image uploaded by the user.

Now let’s see how a setting can actually be used in your HTML code. For example, if you have a setting named **Logo**, you can use it in your HTML by typing `%%SKINSETTINGLOGO%%`, and this will be replaced with the actual value of the setting. For example, Greeny 2 inherits the skeleton skin (which also defines a Logo setting) and the HTML code that uses the setting looks like this:

```
<h2 id="logo">
  <a href="%%URL_INDEX%%">
    
  </a>
</h2>
```

In the actual page this will be replaced with whatever the user uploaded in the Admin Panel.

Following is an example of a Settings section with all 3 types of settings:

```
<Settings>
  <Setting type="image" name="Logo" displayName="Logo" maxWidth="400"
maxHeight="200">customer/images/logo.png</Setting>
  <Setting type="checkbox" name="CheckboxSetting" displayName="Checkbox
setting">>false</Setting>
  <Setting type="text" name="TextSetting" displayName="Text setting">Default
Value</Setting>
</Settings>
```

The usage of image and text settings is really straightforward. The best example for when you would want to use an image setting is for a site's logo image. Then, you would use text settings when you want to allow the site administrator to change a text through the skin settings (although for texts you can also use the `<mi:text>` tag. See below in this chapter). Checkbox settings have only values of 0 (false) or 1 (true). You can use them for example in class names when the value of the checkbox determines whether some element will be visible or not. Here is an example from the Rainbow skin:

```
<Settings>
  <Setting type="checkbox" name="ShowFooterIcons" displayName="Show footer
icons">>true</Setting>
</Settings>
```

The following code illustrates how this setting is used in the skin's footer template:

```
<td id="footer_icons">
  
</td>
```

If the setting was checked, this will result in CSS class "showFooterIcons1", while if it was not checked the CSS class would be "showFooterIcons0". Then the CSS file itself looks like this:

```
td#footer_icons {
  width: 372px;
  height: 184px;
}
td#footer_icons .showFooterIcons0 {
  display: none;
}
```

Skin Settings in Multiple Languages

If you want your skin settings to have different display names for different languages, add a /lang folder under your skin folder (look at the built-in skins for reference). This folder can contain multiple xml files named after the language you are customizing for. In this xml file you can define the display names of your skin settings in the respective language. For example, a bg.xml file that specifies the display name of the Logo setting in Bulgarian will look like this:

```
<?xml version="1.0" encoding="UTF-8"?>  
<Language langCode="BG">  
  <SettingDisplayName name="Logo">Joro</SettingDisplayName>  
</Language>
```

Skin Inheritance

The best practice in skin inheritance is that all skins for the Customer Area must inherit the skeleton skin. This way a skin can only contain the files that it customizes. If a certain HTML template, CSS or JavaScript file was not changed by a certain skin, it should not contain the file, and in this case Summer Cart will use the version from the parent skin. For example, of the 4 skins that are built into Summer Cart, 3 overwrite just the CSS file, and only the Rainbow skin overwrites the header and footer as well. Nevertheless, the sites look very different.

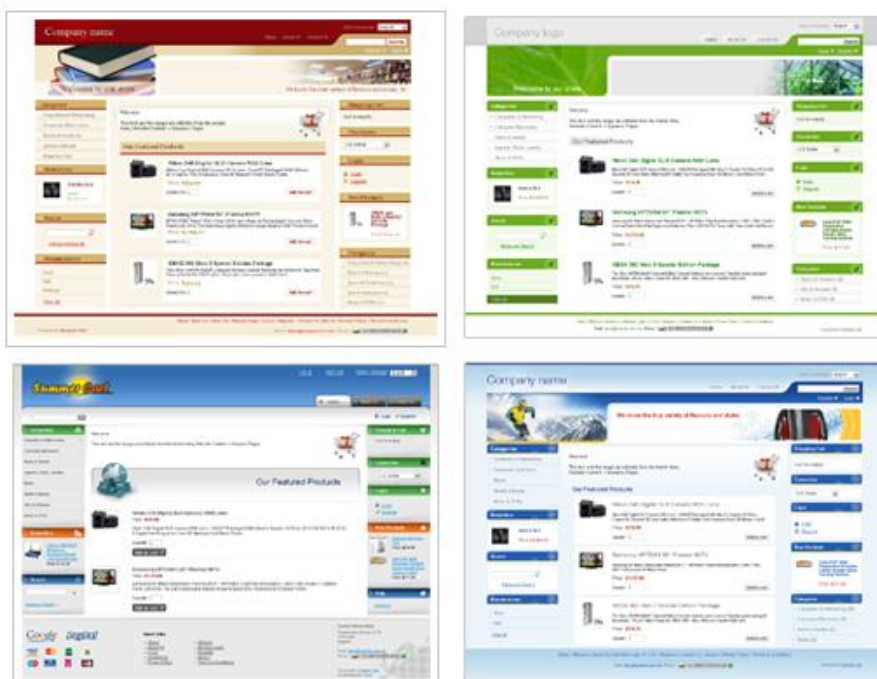


Fig. 2.3 Comparison of skins with purely CSS customizations

Folder Structure

A bare bone skin in Summer Cart needs only its own folder and a skin.xml file. These are the only required elements of a skin, although such a skin will be a complete copy of the basic skeleton skin. Let's look at how the skeleton skin is organized, and you should follow the same folder structure in your skin.

Note that there are several pages (typically ones listing products) where items can be displayed in either a list or a grid, and the customer can switch between the two at any time. These pages have two separate templates – one for the List view (e.g. index.tpl) and one for the Grid view (e.g. index_grid.tpl).

Folder	Description
/customer	Most of the templates for the customer area are located in subfolders of the /customer folder. Itself it contains the page template, header and footer templates, layout area templates, the home page, and a few others.
/customer/addresses	Registered users can have multiple addresses that they can maintain through their profile. This folder contains all the templates for listing, adding and editing addresses.
/customer/categories	Templates for displaying products filtered by category. There are two separate list and grid templates.
/customer/checkout	Templates for each step of the Checkout Wizard, as well as templates for the different results of a checkout – success with processed payment (checkout_success.tpl), success with pending payment (checkout_pending.tpl) or declined payment (checkout_failure.tpl)
/customer/cmsboxes	Templates for the built-in boxes.
/customer/emails	Templates for all the emails that are sent by the system, such as the ones after successful registration or new order.
/customer/forgot_pass	Templates for the Forgotten Password wizard that is typically available from the Login page.
/customer/forms	Templates for forms (forms.tpl) and the confirmation message that is displayed after a form is successfully submitted.
/customer/manufacturers	Templates for manufacturers list (all_manufacturers.tpl), and products filtered by manufacturer. There are two separate list and grid templates.
/customer/myaccount	Templates for pages that only a registered user can access, such as his profile, history of orders, and points. The only exceptions are the templates for the user's addresses, which are in the /addresses folder, and templates for user's wishlist, which are

in the /modules/wishlist folder.

/customer/popup

Templates for popups such as the Send to Friend popup.

/customer/product

Templates for displaying products in a list or a grid.

/customer/register

Templates for the registration page (register.tpl), as well as the page displayed after a successful registration (register_success.tpl).

/customer/search

Templates for displaying search results. Products are displayed in either a list or a grid. The rest of the templates are used for Advanced Search where users can search for a product based on any property or custom attribute of a product.

/customer/widgets

Templates for widgets.

/lang

Language customizations for skin settings. See the chapter Skin Settings in Multiple Languages for further details.

/modules

Templates for modules.

/modules/checkout

Templates for the installed checkout modules.

/modules/news

Templates for the News module: list of news (news_view.tpl), news details (news.tpl) and news box (news_box.tpl)

/modules/payment

Templates for the installed payment modules.

**/modules/product_
comparison**

Templates for the Product Comparison module.

/modules/system

Templates for all the system (internal) modules, such as the modules for bundled products, newsletters, XML and RSS feeds, product ratings & reviews, related products, product prices and wishlists.

Template Structure

Templates in Summer Cart are XHTML files with extension “.tmpl”. Templates can contain various placeholders that are parsed at runtime by PHP scripts to produce dynamic content. You can find all the templates in the system in the skeleton skin, and if you want to customize a certain template file you must copy it over from the skeleton to your own skin in the same folder (e.g. /customer/cmsboxes). Because your skin inherits from the skeleton skin, any file that is missing from your skin’s folder will be loaded from the skeleton skin.

There are two types of dynamic content placeholders in Summer Cart: the %% variables and the <mi:> tags.

The %% variables are of the form %%VARIABLE_NAME%. These will be replaced with actual values at runtime. Most of the %% variables are specific to a page, and the PHP script that is serving the page knows what variables it supports. These you can see in the template you are customizing. There are also variables that are accessible from any page. These are:

Variable	Description
%%SKINSETTING...%%	Allows you to access your skin settings. See the Skin Settings chapter for further details.
%%SKIN_URL%%	Replaced at runtime with your skin’s relative URL. Note that this variable is obsolete, and you should use <mi:skinurl> instead (see next page).
%%CURRENT_LANGUAGE_CODE%%	The code of the language selected by the user
%%HTTPS%%	The base URL of the store using https protocol if the site administrator checked the “Use SSL” option in Settings > Security in the Admin Panel. This is used for pages that have sensitive information such as checkout pages.
%%URL_LOGIN%%	URL of the Login page
%%URL_FORGOT_PASS%%	URL of the Forgot Password page
%%URL_REGISTER%%	URL of the Registration page
%%URL_LOGOUT%%	URL of the Logout page
%%URL_PROFILE%%	URL of the User Profile page
%%URL_ORDERS%%	URL of the User Orders page
%%URL_POINTS%%	URL of the User Points page
%%URL_WISHLIST%%	URL of the User Wish List page
%%URL_ADDRESSES%%	URL of the User Addresses page

%%URL_CHECKOUT%%	URL of the Checkout page
%%URL_VIEW_ORDER%%	URL of the View Order page. This requires Order ID, which is available on pages where the user can select an order, such as the Orders page. For example: %%URL_VIEW_ORDER%%?OrderID=%%ORDERID%%
%%URL_ORDER_EGOODS%%	URL of the E-Goods page. This requires Order ID, which is available on pages where the user has selected an order, such as the Order Details page. For example: %%URL_ORDER_EGOODS%%?OrderID=%%ORDERID%%
%%URL_DOWNLOAD%%	URL to download an E-Good. This requires an E-Good ID, which is available on pages where the user can select an e-good, such as the Order E-Goods page.
%%URL_MANUFACTURERS%%	URL of the Manufacturers page
%%URL_CART%%	URL of the Shopping Cart page
%%URL_SEARCH%%	URL of the Search page
%%URL_INDEX%%	URL of the Index page
%%URL_NEWS%%	URL of the News page
%%URL_PRODUCT%%	URL of the Product Details page. This requires a Product ID, which is available on pages where the user can select a product.
%%URL_SEND_TO_FRIEND%%	URL of the Send To Friend page. This requires a Product ID, which is available on pages where the user can select a product, such as the Products or Product Details page.
%%URL_MODULE%%	URL of the Modules page. The page is used to run a module that you specify by name. For example: %%URL_MODULE%%?ModuleName=com.summcart.rss &UILanguage=%%CURRENT_LANGUAGE_CODE%% &FeedType=News

The **<mi:> tags** are of the form `<mi:tagName>...</mi:tagName>`. The main difference between the `<mi:>` tags and the `%%` variables is that the tags have an HTML value inside of them, and at runtime this value is processed to form the end result. With the `%%` variables the PHP script just replaces the variable with the generated dynamic content. Tags supported by Summer Cart are:

Tag	Description
<code><mi:skinurl>URL</mi:skinurl></code>	Converts the given URL to one that is relative to your skin folder. The main benefit over the <code>%%SKIN_URL%%</code> variable is that if the URL is not found inside your skin, it will be looked for in its parent skin. This way if you need to customize a single template you do not need to copy over from the parent skin all the base files that your template references.
<code><mi:text>text</mi: text></code>	Summer Cart is a platform that supports multiple languages. Typically, you would want any text that appears on the Customer Area to be localizable. For this to work you should use <code><mi:text></code> tags, and inside them put your original text (typically that would be the text in English). Then when you go to Admin Panel > Settings > Languages, all these tags in your templates are extracted and displayed in a list. The store owner can install a language pack or hire a translator to go through the list and translate the texts, and then the localized versions will be displayed to the end user based on his language preferences.
<code><mi:section>...</mi:section></code>	These are dynamic sections of HTML, much like a template within a template. Each section has a name, and this is how the PHP script that parses the template finds the section. Most sections in Summer Cart are inline, which means that their content is placed directly between the opening and the closing tags. Some sections have a <code>src</code> attribute, and these have their contents loaded from an external file. Basically you should remember to keep all the sections in place, because the PHP script uses them to show or hide certain HTML code, as well as to repeat certain HTML code multiple times, for example with product listings. Note: If you copied a template from the skeleton to your skin, and that template has sections with their content loaded from other files, you must copy these other files to your skin as well.

HTML and CSS Structure

The content of each Summer Cart page is generated from a number of templates as illustrated on fig. 2.4. First, the header template is processed, then the top area, the left area, the page template (actual page content is rendered inside the page template), the right area, the bottom area and finally the footer template. All these templates are located in the /customer folder.

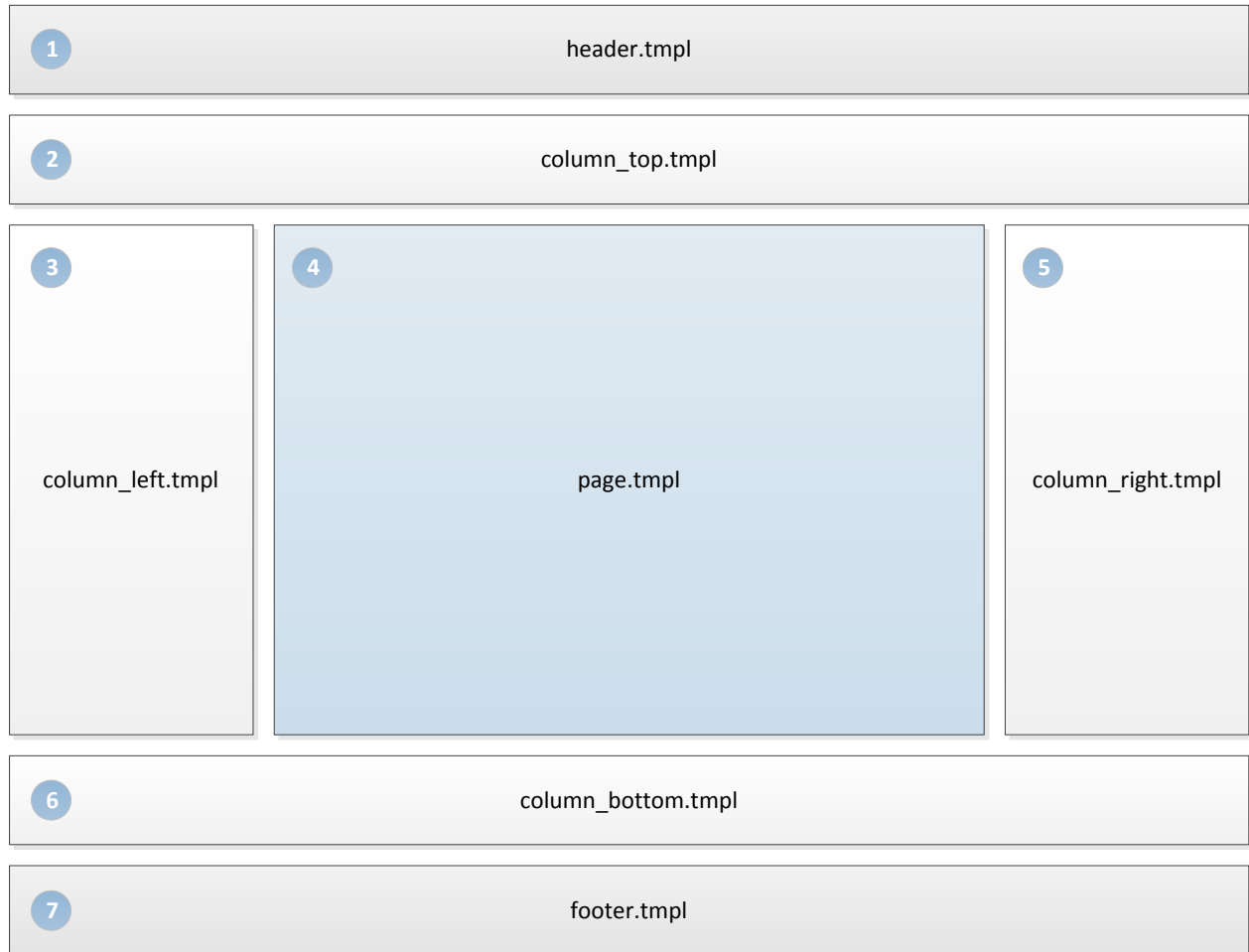


Fig. 2.4 Illustration of what templates are used to build a page

The CSS files for a skin are located in its /css folder. A typical skin would have all of its CSS rules defined in a main.css file, and the other css files in the folder would be related to 3rd party libraries (e.g. jquery.css, colorbox.css) or browser-specific customizations (e.g. ie6.css).

Now let us examine the HTML and CSS structure of a typical Summer Cart page, in this case the Home page. Note that the common elements can be referenced by ID, and some of them also have CSS classes so you can reference groups of elements by common attribute. For example, all horizontal areas (the top and bottom areas) have the “h-column” class, while the vertical ones (the left and right areas) have the “v-column” class, although all four areas can be referenced by their ID as well.

```
<html>
  <head>
    ...
  </head>
  <body class="lang-en dynamic-page dynamic-page-index">
    <div id="body-container">
      <div id="container">
        <div id="header">...</div>
        <div id="column-top" class="h-column clearfix">...</div>
        <div class="mainWrapper">
          <table id="main">
            <tr>
              <td id="column-left" class="v-column">...</td>
              <td id="content">...</td>
              <td id="column-right" class="v-column">...</td>
            </tr>
          </table>
        </div>
        <div id="column-bottom" class="h-column clearfix">...</div>
        <div id="footer" class="clearfix">...</div>
      </div>
    </div>
  </body>
</html>
```

The Body of each page uses a number of CSS classes. What classes are used depends on whether it is a Dynamic or Rich Text page, and whether additional CSS classes are specified in the layout that the page is using. One common CSS class that is set on all pages is the language class. In the example above, it is “lang-en”. What language class will be used depends on what language the customer selected in the Customer Area. The available languages are set up in Admin Panel > Settings > Languages. Each language has a Language Code setting, and it is used in the class name (e.g. the code for English is “en”, so the language class name is “lang-en”). This gives you the option to apply CSS rules for a specific language only.

Dynamic pages’ bodies have a language class, as well as the “dynamic-page” and “dynamic-page-...” classes (“...” being the respective page name). In addition they have the CSS classes inherited from the page’s layout.

```
<body class="lang-en dynamic-page dynamic-page-index">
```

All dynamic pages have their own ID’s, e.g. “index-page”, “login-page”, etc. Some of the pages have 2 different views – List view and Grid view, and they have two separate templates. For example, the Home page has two templates: “index.tpl” and “index_grid.tpl”, and the

customer can switch between the two at any time. The container divs of the two templates use different CSS classes.

```
<div id="index-page" class="list-page">...</div>
```

The element above is from `index.tpl`, while the one below is from `index_grid.tpl`.

```
<div id="index-page" class="grid-page">...</div>
```

Rich Text pages' bodies have a language class, as well as classes inherited from their layout.

```
<body class="lang-en">
```

Boxes have 2 classes: "box" and "box-#", where # is the box ID in Summer Cart. This way you can write CSS rules that target specific boxes, even custom HTML boxes created in the Admin Panel. Each of the built-in boxes also has its own HTML ID (e.g. "cart-box", "login-box", etc.). Boxes typically contain two divs, one with the box title, and one with the box content.

```
<div id="cart-box" class="box box-%%CMSBOXID%%">
  <div class="box-title-wrapper">
    <h2 class="box-title"><mi:text>Shopping Cart</mi:text></h2>
  </div>
  <div class="box-content">
    ...
  </div>
</div>
```

When you create custom HTML boxes from the Admin Panel, you specify the box title and content. In this case the "cms_html_box.tpl" template is used from `/customer/cmsboxes`. These boxes have the "html-box" CSS class. From the Admin Panel you can change the visibility of the title div, and so a `<mi:section>` tag is used around the box title div to show or hide it.

```
<div class="box box-%%CMSBOXID%% html-box">
  <mi:section name="cmsHtmlBoxHasTitle">
    <div class="box-title-wrapper">
      <h2 class="box-title">%%CMSHTMLBOXTITLE%%</h2>
    </div>
  </mi:section>
  <div class="box-content">%%CMSHTMLBOXCONTENT%%</div>
</div>
```

There are several boxes that have similar content. For example, the Bestsellers box and the New Products box both show a list of products. These boxes have common CSS classes, so you can reference all of them with a single CSS rule.

```
<div id="bestsellers-box" class="box box-%%CMSBOXID%% products-box">
  ...
</div>
```


On numerous places in the templates, you will see 2 or 3 divs being placed in one another. This is so various desired effects, such as round corners, can be defined purely in CSS, without the need of changing the HTML code.

```
<div class="page-content">
<div class="page-content-bg">
<div class="page-content-bg-repeat">
...
</div>
</div>
</div>
```

Layouts can be targeted by setting the “CSS Class Names” field in the layout details with a list of comma-separated CSS class names. Note they will be added directly to the document body classes.

Layout areas can be targeted by their ID (e.g. “column-top”, “column-right”) as well as by their orientation (e.g. “h-column”, “v-column”).

```
<div id="column-top" class="h-column clearfix">...</div>
```

Menus are targeted based on location. The main menu can be targeted using the “main-menu” class, and the fast menu using the “secondary-menu” class. Both menus are displayed in the header of the site, and therefore their HTML template can be found in header.tpl. The footer menu can be targeted using the “footer-menu” class, and its HTML is in the footer.tpl file. Boxes that contain menus will have the “box-menu” class, and you can target a specific menu using the box ID (e.g. “box-23”).

Titles (headers) are typically used inside two divs, so you can easily customize all titles (using the “title-wrapper” class), titles with specific level (“h1-title”, “h2-title”, etc.), or titles with specific alignment (“title-left”, “title-right”).

```
<div class="h1-title title-wrapper">
  <div class="title-left">
    <h1 class="title-left">Contact</h1>
  </div>
</div>
```

Breadcrumbs are generated inside two divs, “breadcrumb” and “breadcrumb-bg”, and the links are put into spans. This way you can easily put icons between breadcrumb links. The active link would have the “active” class.

```
<div id="breadcrumb">
<div class="breadcrumb-bg">
  <span>
    <a href="/index.html">Home</a>
  </span>
  <span>
    <a href="/category/7/game-consoles.html">Game consoles</a>
  </span>
```

```
<span class="active item fn">Sony Playstation 3 120GB Slim Console</span>
</div>
</div>
```

Links are best targeted as a whole, or per specific page or box using page and box-based CSS selectors.

Inputs can be targeted as a whole (e.g. `input[type="..."]`) or using the two common classes, “input-text” for textboxes and “input-submit” for buttons. Throughout the templates, buttons are used in their own span element with class “button”. This allows for various effects purely through the CSS, such as expandable buttons with round corners.

```
<input type="text" class="input-text" name="Email" id="Email" .../>

<div class="button-container">
  <span class="button">
    <input class="input-submit" type="submit" value="Submit">
  </span>
</div>
```

Finally, there are a number of common CSS rules that are used in many templates. These rules are listed near the end of the main.css file so they have priority over other rules. Following is a list of these common rules:

```
.left {float: left;}
.right {float: right;}
.right, th.right {text-align: right;}
.left, th.left {text-align: left;}
.center {text-align: center;}
.inline {display: inline;}
.wide {width: 100%;}
.clear {clear: both;}
.clearfix {display: block;}
.hidden {display: none;}
```

Code Style and Conventions

Here we will discuss the code style we are following in Summer Cart and the conventions we are using. Although you are not required to follow the same style and conventions, you are strongly advised to do so, as your code will be easier to read and maintain.

For HTML, use lower-case HTML tags and double quotes.

```
<input class="input-submit" type="submit" value="Submit">
```

Make sure you read and understood the HTML Structure and CSS Structure chapters, and you follow the same structures for your templates.

For CSS, keep your CSS rules in a single file called “main.css”, and keep 3rd party CSS rules (e.g. jquery.css) and browser customizations (e.g. ie6.css) in separate CSS files. Use lower case class names separated by dashes. Keep CSS rules on single lines.

```
#main-menu {float: right; margin: 0; padding: 0; list-style: none none;}
#main-menu li {float: left; margin: 0 15px 0 0; padding: 0 0 0 0; list-style: none none;}
```

CSS rules are organized into sections using `/* Section */` comments. Sections would typically be named after the element being customized. For example, the CSS rules for the Login box will be listed after a `/* Login box */` comment.

```
/* Login box */
#login-box a {float: left; width: 80px;}
#login-box a.wishlist {margin: 0;}
```

How Do I...

Customize the CSS of a site through the Admin Panel?

You can't. To modify the CSS of a site you have to create a new skin (or copy an existing one). A basic skin that only customizes the CSS contains only a few files and can be created in a few minutes. Read how in the Skins section.

Set CSS rules for a specific layout?

Layouts have a property called "CSS Class Names". Whatever you type here will be the CSS class of the body of pages that use this layout.