PRINTSHOP MAIL SUITE

PrintShop Mail Web
Skinning Guide

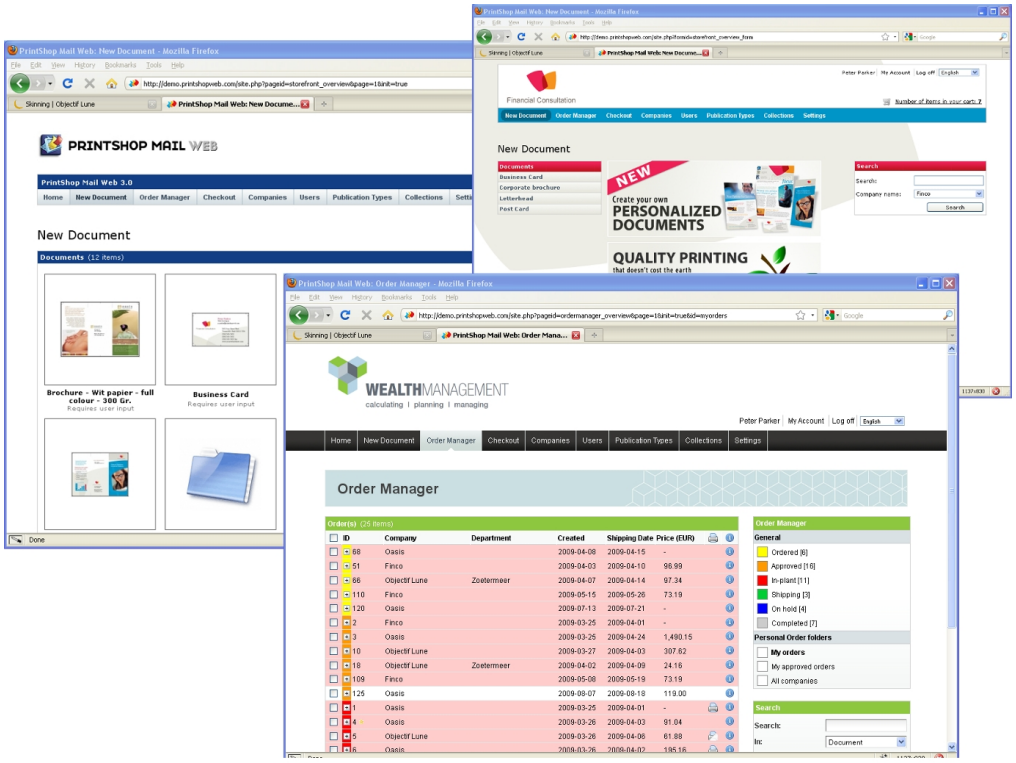
Copyright Information

Title	PrintShop Mail Web - Skinning Guide
Revision	2010/01/04

Table of contents

1 Introduction

The PrintShop Mail Web website is fully *skinnable*, everything except the actual textual content of the page can - and will - vary from skin to skin. A skin is a series of files that control the presentation of the web site. To allow PSM Web to be skinnable, style is completely separated from contents. The web page use standard HTML elements, user defined classes and unique IDs. The style information is stored in Cascading Style Sheets documents, allowing the styles to be changed and manage the styles without the need to change the source code.



PrintShop Mail Web skin examples

2 Getting started

Creating a skin for the first time can be a daunting and sometimes a time consuming task. This section gives you an overview of useful tools and sources on the web.

This section answers the following questions:

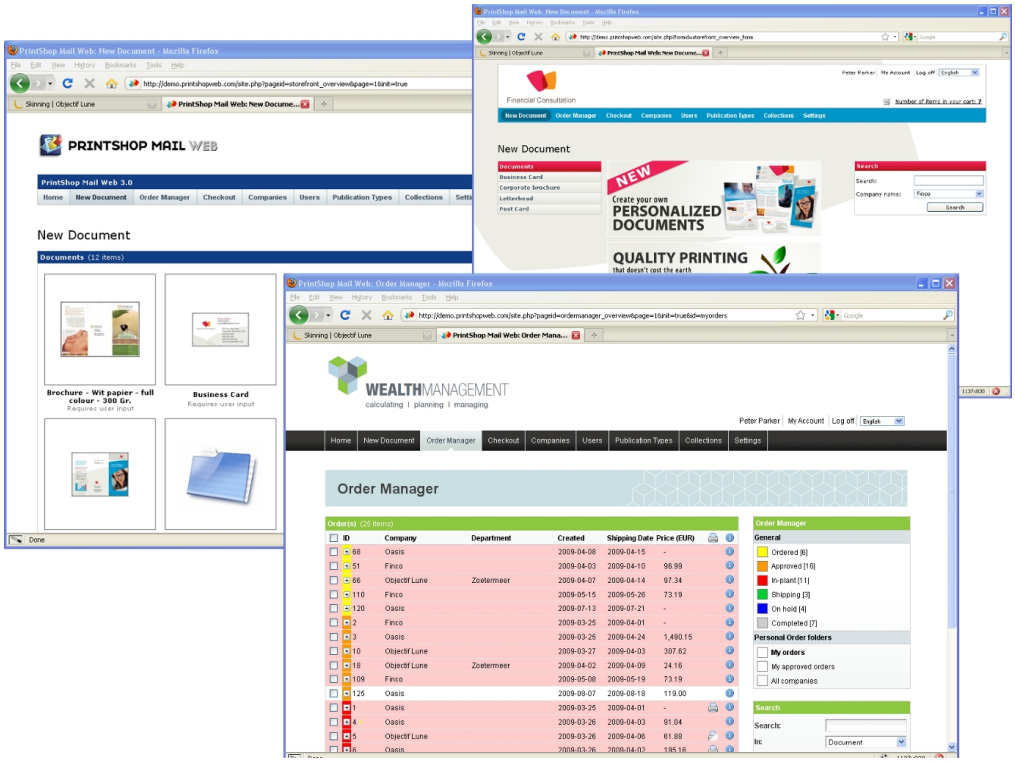
2.1 Key Concepts

In order to create customs skins you should understand the following key concepts:

- Skinning (Page 3)
- Templates and styles (Page 4)
- File organization (Page 4)

2.1.1 Skinning

The term *Skinning* refers to a software architecture which provides you with a manageable way of separating application logic and content from its presentation. This abstraction of *form* and *function* is sometimes referred to as a two-tiered presentation model. This allows web designers and system integrators to customize the user interface without the risk of interfering with the functionality of the application. Skins are used to define colors, fonts, images, borders and the placement of various components in the web site.



PrintShop Mail Web skin examples

PSM Web can use a different *skin* for each company, allowing your customers to use the system in their own house style or a style that closely matches their house style (depending on the time and effort you have spend to match there corporate site). Users of that company can access the PSM Web web site via a personalized URL which will invoke their skin. Personalized URL is controlled by the PSM Web administrators by defining an URL variable for each company.

For example: http:/www.yourprintshopmailwebsite.com/yourclientscompanyname.

2.1.2 Templates and styles

Skinning involves the use of templates. A template is a series of files within the PSM Web web site that control the presentation of the content and their position on the page. An embedded skinning engine is used to assemble the dynamically generated content into the final HTML pages.

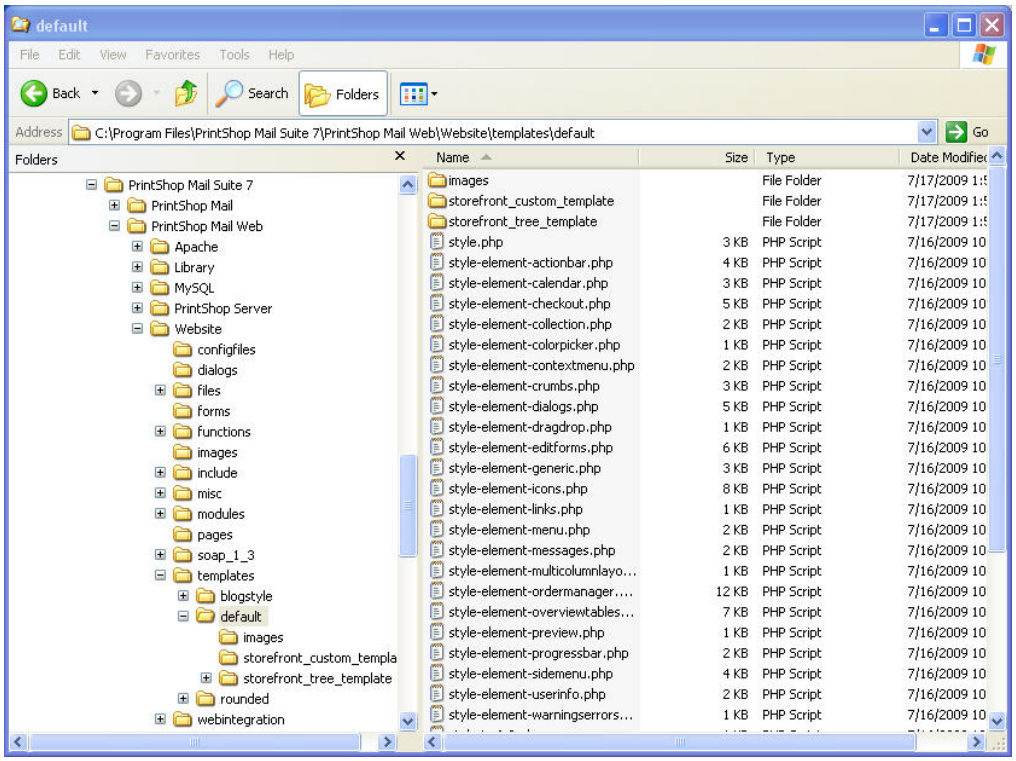
Skins are used in combination with *Cascading Style Sheets* (CSS). This is a mechanism for adding style (e.g. fonts, colors, spacing) to Web documents, standard by the World Wide Web Consortium (W3C). A skin combines uses HTML, JavaScript and PHP to define the look & feel of PSM Web. To create a new skin or modify an existing skin some experience with these techniques is necessary.

2.1.3 File organization

Skins are organized under the *Templates* folder located in the PSM Web web site folder. Each skin has its own directory tree. The name of this folder will be visible in the *Skin*-pull down menu in the Web Design section of a company and in the web design page in the *Settings*-menu.

Skin folders stores the items required to render a skin and can contain an unlimited number of files and sub folders. A skin consists of the following elements:

- Template.php
- Cascading Style Sheet documents
- Images
- Web design parameters stored in the PSM Web database



The contents of skin folder

Template.php

The *Template.php* file is the main file for your skin and can be compared with an index.html file of a regular HTML web site. The file calls the different functions of the skinning engine that assembles the final HTML output. The *Template.php* file controls the logical positions of the page elements and processes the items in the order that they appear in that file.

Style.php

As stated before skins are used in combination with Cascading Style Sheets (CSS). PSM Web uses a PHP file that generates a CSS structure. To be able to call special functions that insert special variables like color information or an image file. The variable information is retrieved from the PSM Web database and can be set by the administrator in the *Edit Web Design* page. This can be done per company (f.e. to match the companies color scheme) or for the system wide web design settings.

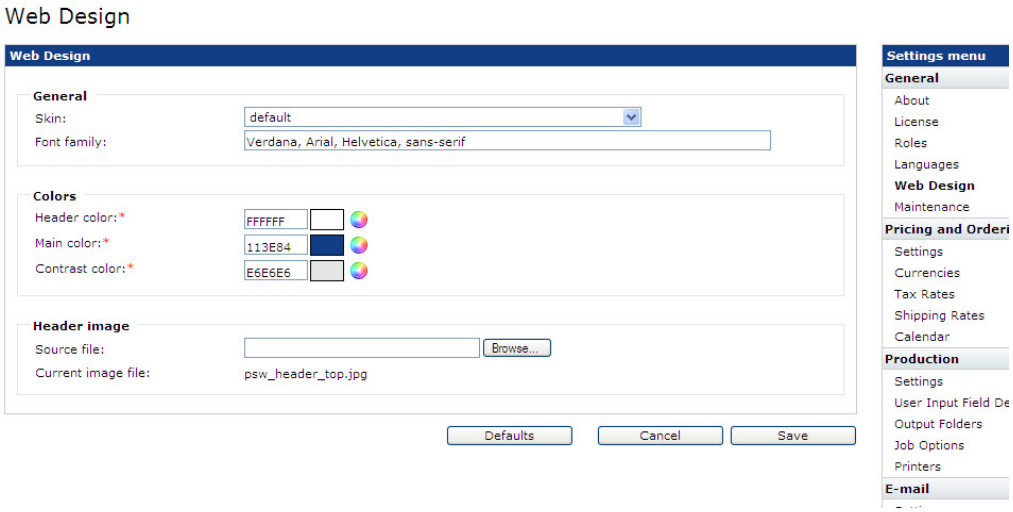
PSM Web comes with a browser detection mechanism. This allows you to make style exceptions for specific browsers, browser versions and operation systems.

Images

To enhance your skin even more you might want to add custom images and icons. You can place these images inside your skins folder and refer to them from within your style documents.

Web design parameters

By editing the web design settings of a company you can specify company specific colors, fonts and logo. Using special functions a skin can retrieve these parameters and use them in the *Style.php* file.



The Edit Web Design page

2.2 Creating your own skin

Before creating your own skins you should have some basic knowledge of HTML, CSS, PHP and Javascript.

2.2.1 Resources

There are fine books on these subjects and you will find a lot of information on the web. The web is probably your best source. If you're not familiar with HTML, CSS and Javascript please visit the following web sites:

- http://www.w3schools.com
- http://www.htmldog.com
- http://jquery.com

2.2.2 Tools

What is a craftsman without a good set of tools. Below you'll find a list of tools that will help you with skin development.

PrintShop Mail Web demo version

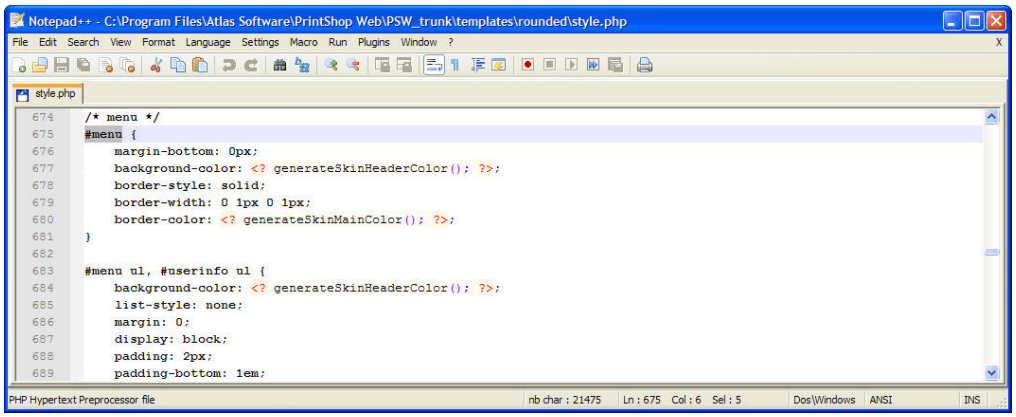
Download and install the demo version of PSM Web on your local machine. This allows you to develop your skins offline without the risk of interfering with your production environment. The demo version is limited regarding printing and previewing PrintShop Mail documents but has a fully functioning user interface. A demo version can be downloaded from the Objectif Lune web site (www.objectiflune.com), PrintShop Mail Web is part of the PrintShop Mail Suite.

Design

Before you start creating a skin you should have a visual theme which determines the look & feel of your web site. This can be the design of your current corporate site, your customers web site or a new design. You'll probably start your design in an application like Adobe Photoshop ending up with a bunch of sliced images (background images, icons, button backgrounds etc). On the web you can find several samples and downloadable .psd files which will get you up and running in no time.

Text editor

There are several specialized CSS editors available. The PSM Web style sheet documents are a combination of CSS and PHP so we advise you to use a text editor like Notepad++. This is an open source editor (a Notepad replacement) that support coding in several programming languages. The application runs under the MS Windows and can be downloaded for free at the following location: http://notepad-plus.sourceforge.net/

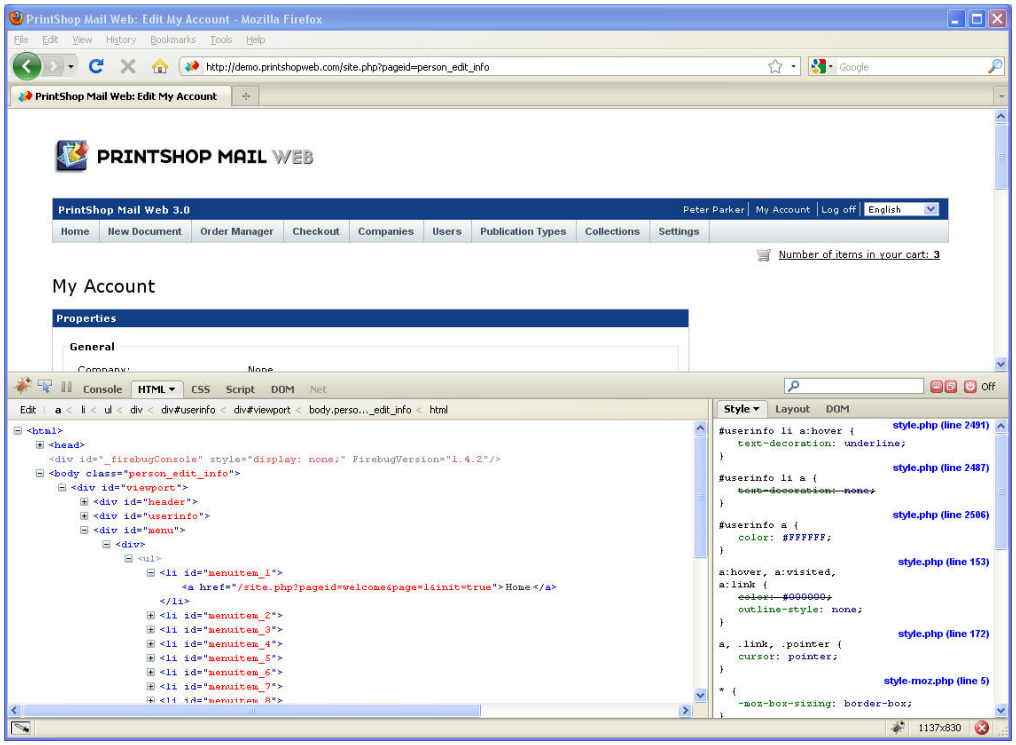


The style.php file in Notepad++

Browser extensions

The toughest part will be to keep your skin compatible with the various browsers and browser versions. You can download and install additional browser extensions that will help you during skin creation. Typically these extensions expose the hierarchical structure of your web page allowing you to identify tag names, ids and CSS classes quickly. These extensions are available for all popular web browsers and can be downloaded for free. Below you'll find a list of popular extensions:

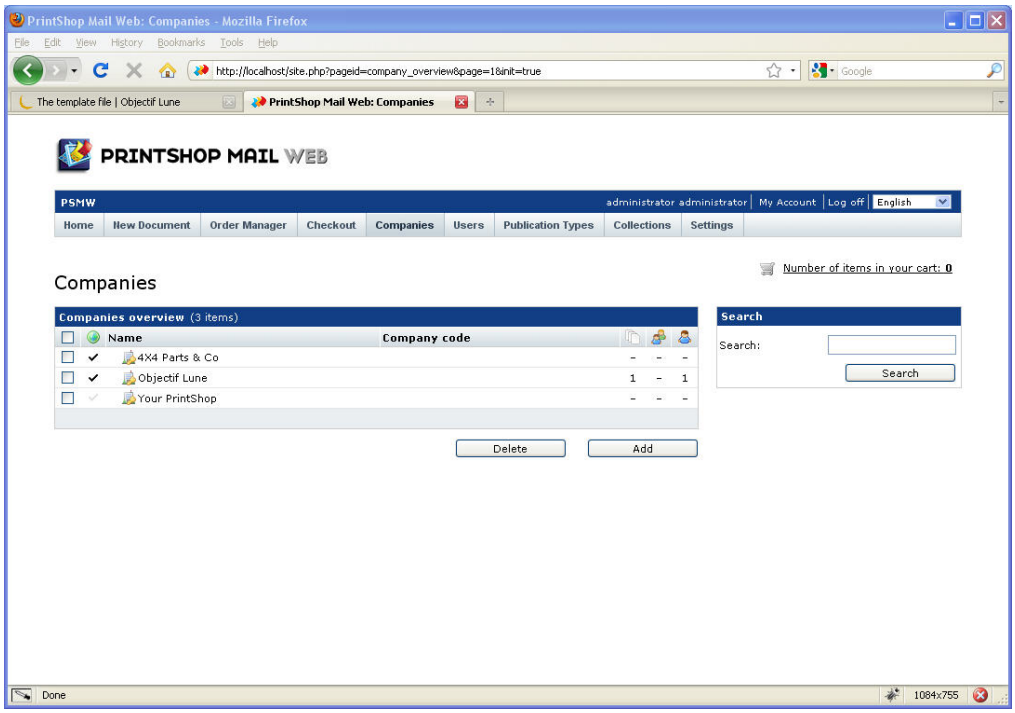
- Firebug for Fire Fox (our personal favorite)
- Microsoft Internet Explorer Developer Toolbar



The HTML DOM exposed by the Firebug extension for Fire Fox

3 The template file

The Template.php file contains the main outline of the PSM Web web site, it contains HTM code with a few snippets of PHP. It is invoked for each page request and outputs the final page contents. The PHP snippets control the rendering of PSM Web function blocks on a page by page basis. Examples of functions blocks are: the menu bar, sub menus, the user info bar, the main content area.



A sample PSM Web web page showing various function blocks

3.1 HMTL Outline

The *Template.php* file contains regular HTML code combined with PHP. The main structure of the file contains the standard HTML elements like <html>, <head> and <body>. Custom HTML code can be added to insert information/content to the pages and to change the positioning of the elements.

The design and positions of the function blocks can be changed by altering the *Style.php* file or by adding inline styles.

Typically function blocks are placed inside HTML tags. The following sections describe the main function blocks. Please note that the structure of the *Template.php* will probably be different for each skin. The following code snippets are derived from the default PSM Web skin.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title><? generateSystemName(); ?>: <? generateTitle(); ?></title>
  <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon" />
  <? generateCSSIncludes(); ?>
  <? generateJavaScriptIncludes(false); ?>
</head>
<body class="<? generatePageID(); ?>">
  <div id="viewport">
  <div id="header"><h1>Hello World</h1></div>

  <? if(function_exists("generateUserInfo")) { ?>
    <div id="userinfo">
      <div>
        <h3><? generateAdditionalInfo(); ?></h3>
        <? generateUserInfo(); ?>
      </div>
    </div>
  <? } ?>

  <? if(function_exists("generateMenu")) { ?>
    <div id="menu">
      <div>
        <? generateMenu(); ?>
      </div>
    </div>
  <? } ?>

  <? if(function_exists("generateCrumbs")) { ?>
    <div id="crumbs">
      <? generateCrumbs(); ?>
    </div>
 <? } ?>

  <? if(function_exists("generateMessages")) { ?>
    <div id="messages">
      <? generateMessages(); ?>
    </div>
  <? } ?>

  <div id="content">
    <? if (getFormCount() > 0){ ?>
      <? generateContent(); ?>
    <? } ?>
```

```
  </div>

  <? if(function_exists("generateSubMenu")) { ?>
    <div id="submenu">
      <? generateSubMenu(); ?>
    </div>
  <? } ?>

  <? if(function_exists("generateSubMenu1")) { ?>
    <div id="submenu1">
      <? generateSubMenu1(); ?>
    </div>
  <? } ?>

  <? if(function_exists("generateSubMenu2")) { ?>
    <div id="submenu2">
      <? generateSubMenu2(); ?>
    </div>
  <? } ?>

  <? if(function_exists("generateSearch")) { ?>
    <div id="search">
      <? generateSearch(); ?>
    </div>
  <? } ?>

  <? if(function_exists("generatePreview")) { ?>
    <div id="preview">
      <? generatePreview(); ?>
    </div>
  <? } ?>

  <? if(function_exists("generateSummary")) { ?>
    <div id="summary">
      <? generateSummary(); ?>
    </div>
  <? } ?>

  </div>
  <div id="footer"></div>
</body>
</html>
```

As you can see the PHP snippets perform a *function_exists* check. You should not alter these checks but you may add HTML tags and information inside or outside these checks. To meet your design you will probably need to add additional and/or <table> tags to change the positioning of the function blocks.

3.1.1 Page title

The <title> tag is part of the <head> and contains the name of the web page. The contents of this tag are shown in the browser title bar - at the very top of the window. The title bar shows the name of your browser and the name of the page. When a visitor adds your page to their favorites or bookmarks, the page title is the default name it will assign to the bookmark. Search engines use the page title as the listing name for their results.

You can enter a custom title or use the predefined PHP functions to set the title dynamically. By default the <title> tag will be populated with the systems name and the title of the current section/subsection. These two values are separated by a colon as follows:

```
<title><? generateSystemName(); ?>: <? generateTitle(); ?></title>
```

The *generateSystemName* returns the value of the *cSystemName* string and depends on the selected language. You can change the value for this string in the *Edit Language* page of the *Settings* section. The *generateTitle* will return the name of the current PSM Web section.

3.1.2 CSS includes

This function retrieves the proper location of your skin and its accompanying CSS files. This PHP function is required to dynamically load your CSS files. You should not remove this.

```
<? generateCSSIncludes(); ?>
```

3.1.3 Header

The information stated in the header div is used to display the main page header or title (not to be confused with the browser window title). Using the *Style.php* you can replace the contents of this element with the logo uploaded through the *Edit Web Design* page. It allows the skin to show a company specific text or image.

```
<div id="header"><h1>Hello World</h1></div>
```

3.1.4 User information

This function will add the *User Info* block to your site. This block shows the name of the logged on user, an option that lets the user edits his or her personal information and the *Log off* option. The *My Account* option is optional and depends on the role of the logged on user.

In the default the user User Info block is dark blue and placed just above the menu bar

In the picture above additional text is added to the user info bar (the text PSM Web). You may hard code this text, in this case it is retrieved using the *generateAdditionalInfo* function. The returned value depends on the selected language and can be changed by editing the *cAddiontalInfo* string in the *Edit Language* page (*Settings*).

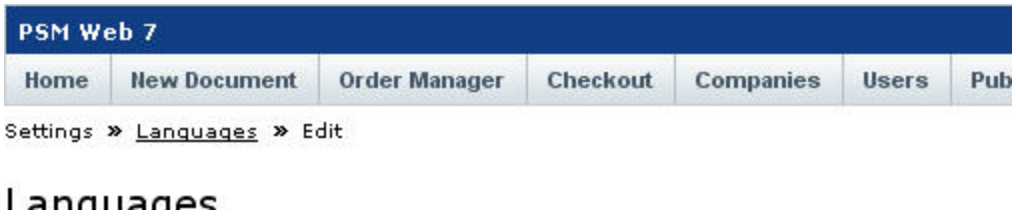
3.1.5 Menu

This function generates the main menu bar.

In the default skin the Menu bar is the part below the User Information bar

3.1.6 Crumbs

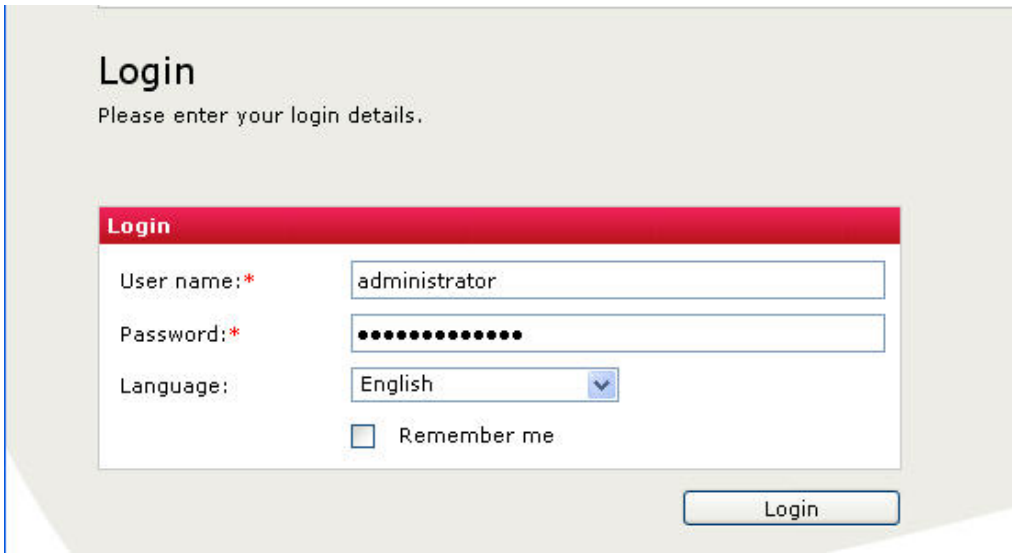
On a Web site, a bread crumb trail is a navigation tool that allows a user to see where the current page is in relation to the Web site's hierarchy. The term bread crumb trail comes from the story of Hansel and Gretel, who left a trail of bread crumbs as they walked through the forest so they could trace their way back home.



Bread crumbs in action

3.1.7 Messages

The *generateMessages* inserts the main headline and descriptive text for each page.



The login page showing a headline and descriptive text

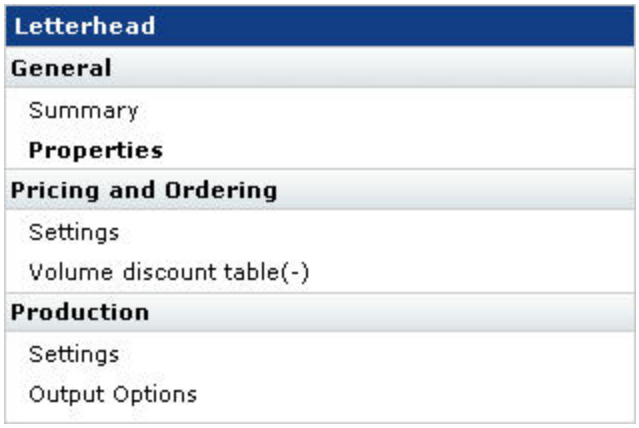
3.1.8 Content

The *generateContent* function renders the main area. It holds the overview information, forms, buttons and the main process functionality of the PSW system. At the *Login* page shown previously the login details and the *Login* button are part of the information rendered by the *generateContent* function.

3.1.9 Sub menus

Sub menus are uses throughout the PSM Web web site. A good example is the *Settings* sub menu and sub menus inside the *Publication Types* and *Companies* sections. These sub menus typically show a list of items that can be used to navigate to a sub item of that section. Using the Style files you will be able to alter the look of these list items, for example changing the color of the list item when you place the mouse over the item (hover).

Because some sections have multiple sub menus, there are multiple *generateSubmenu* blocks in the template file.



A part of the company section showing the Company sub menu

3.1.10 Search

Many overview pages have a search option. The *generateSearch* functions renders this block depending on the section the contents of this block can change.

3.1.11 Summary

The summary is a special block in the *Order Manager* that shows summary information regarding the orders in a specific order folder.

3.1.12 Preview

The *generatePreview* function renders the sidebar items that hold preview information. This can be the part of the site that hold the thumbnail when previews are requested or hold the thumbnail information and download link in the *Order Manager* section.

3.1.13 Footer

This functions lets you render a site wide footer. You may also place static information in this area to state your address and Internet information.

4 Styling

The style files hold the styling parameters for your skin. It defines the fonts, colors, borders, backgrounds, width and height specifications of the user interface elements.

This section contains answers to the following questions:

4.1 Key Concepts

To apply styling to HTML elements, you should understand the following key concepts:

- Tags, IDs and classes (Page 16)
- CSS Syntax (Page 16)

4.1.1 Tags, IDs and classes

Each element on the page is rendered using standard HTML tags or by using and tags that have a unique identifier and/or a class.

IDs are used when there is only one occurrence on a page, classes are used when there are one or more occurrences on a page. CSS IDs are similar to classes in that they define a special case for an element. In other words, they assign an identifier. Standards specify that any given id name can only be defined once within a page or document.

You can reveal ID and class information by viewing the source of the web page or by using additional web developer browser extensions. Your style files will need to refer to these elements to set their formatting.

```
<div id="fcCompanyNameLine" class="formLine ">
 <span id="fcCompanyNameLabel" class="formLabel">Company name:
   <span class="asterisk">*</span>
 </span>
 <span class="formElement">
   <input type="text" value="Oasis" name="fcCompanyName"
    id="fcCompanyNameField" class="formField"/>
 </span>
</div>
```

4.1.2 CSS Syntax

The CSS syntax is made up of the following parts: a selector, a property and a value:

```
selector {property: value}
```

Normally the selector is the HTML element/tag you wish to style, the property is the attribute you wish to change, and each property can take a value. A colon is used to separate the property and its value. They are surrounded by curly braces, see the snippet below:

```
h1 {color: lime}
```

You can specify more than one property, the properties should be separated by a semicolon.

```
fieldset {
 width: 70%;
 margin-left: 1.25em;
}
```

Selectors can be grouped by separating each selector with a comma.

```
a:hover, a:visited, a:link{
 color: black;
 outline-style: none;
}
```

With the class selector you can define different styles for the same type of element. The class selector is defined using a point.

```
.totalprice {
 font-weight: bold;
}
```

You can also define styles for HTML elements using the id selector. The id selector is defined as a # followed by the id.

```
#tableOverview td {
  padding: 2px;
  border-width: 0 0 1px 0;
  border-color: black;
  border-style: solid;
}
```

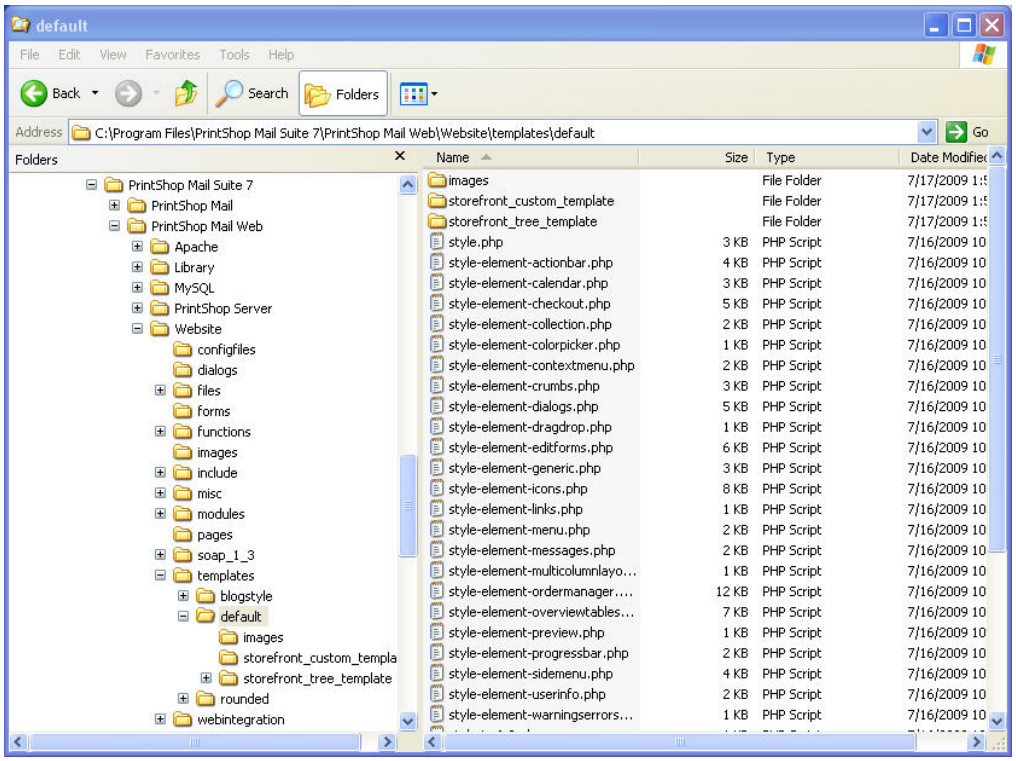
You can insert comments in CSS to explain your settings. Like HTML comments, CSS comments will be ignored by the browser. A CSS comment begins with "/*", and ends with "*/".

```
/* Submenu and Search items start here */
#search, #submenu, #submenu1, #submenu2, #preview, #summary {
 width: 100%;
}
/*
multi-line
comment here
*/
```

4.2 Style organization

The styles that come with PSM Web are organized using the following structure:

- Global styles (Page 18) (generic interface elements)
- Page specific styles (Page 19) (exceptions)
- Browser exceptions (Page 19)



The contents of skin folder

4.2.1 Global styles

The *style.php* file and the *style-element-<subject>.php* files hold the style definitions that apply to all pages in the system. At the beginning of the file the *skin_customization.php* file is included using the PHP *include_once* function. The *skin_customization.php* allows you to use web design parameters like colors and fonts that can be specified per company (*Edit Web Design*). Please refer to the *Special Variables* chapter for more information regarding this option.

```
<?php include_once "../../functions/skin_customization.php" ?>
```

At the end of the *style.php* file the separate *style-element-<subject>.php* and page specific exception files are included. These documents hold the item specific and page specific style exceptions. You could also add your exceptions to the main style file.

```
// Element/Item Specific CSS includes
include_once "style-element-actionbar.php";
include_once "style-element-editforms.php";
include_once "style-element-calendar.php";
include_once "style-element-colorpicker.php";
```

```
...

// Page Specific Items
<? include_once "style-pagespecific.php" ?>
```

4.2.2 Page specific styles

Each page comes with a body tag that has a unique class (there can only be one body tag in a HTML page). This class set on the <body> tag allows you to create page specific exceptions. In the skins supplied with PSM Web the page specific styles are stored in the *style-pagespecific.php* file. This file is included at the end of the *style.php* file.

```
/*
** The width of the required date field is changed for
** the order_information and order_edit page
*/

body.order_information #fdRequiredDateField,
body.order_edit #fdRequiredDateField
{
 width: 10em;
}
```

4.2.3 Browser exceptions

To solve browser specific rendering issues you can create style exceptions for a specific browser or browser version. Below a snippet from the *style-moz.php* exception file (FireFox).

```
/*
** Mozilla specific style sheet entries
*/

* {
 -moz-box-sizing: border-box;
}

.formLabel, .downloadLineLabel, .formStatic, .fieldComment, .formStaticUnits,
.formStaticPrice, .formLink, .formStaticFlex, .formSelect, .formField,
.formFieldNoWidth, .progressLabel, .progressBarWrapper, .progressBarItemDone,
.progressBarItemTodo, .warningLabel, .warningIcon, .warningItem
{
 display: table-cell;
}
```

Search order

The search order is as follows:

1. browsername
2. browsername-version
3. browsername-os
4. browsername-os-version

Browser names

The browser names use the following abbreviations:

- ie = Internet Explorer
- moz = Mozilla
- saf = Safari

Operating systems

The operations system names use the following abbreviations:

- nt = Windows XP
- nt = Windows 2003 Server
- nt = Windows NT
- mac = Mac OS X

Note: Exceptions made for FireFox are stored in a moz exception file. When making an exception for a specific version of FireFox the version number should refer to the Mozilla engine, this number differs from the FireFox version number.

4.3 Header

The information stated in the header div is used to display the main page header or title. This is the place to show a system wide logo or the logo of your customer.

The default skin showing the PSM Web logo as the header image

4.3.1 Replacing content

The HTML snippet below shows the header div containing a *<h1>* tag. Using the *Style.php* you can replace its contents by a logo uploaded in the *Edit Web Design* page. This allows the system to show a company specific image or logo.

```
<div id="header"><h1>PrintShop Web</h1></div>
```

The first style sets the width and height for the header div and sets a background image. The *generateSkinLogo* function (PHP) retrieves the path of the company logo which can be set in the *Edit Web Design* page. Of course this can be replaced by a hard coded path a custom image. The *generateSkinHeaderColor* retrieves the header color specified in the *Edit Web Design* page. This color can be used to fill up the background to match the color of your background image.

```
/* header */
#header {
 height: 70px;
 width: 100%;
 background-image: url(<? generateSkinLogo(); ?>);
 background-position: center left;
 background-repeat: no-repeat;
 background-color: <? generateSkinHeaderColor();?>;
}

#header h1 {
 display: none;
}
```

The second style hides the contents of the <h1> tag (display: none;).

4.4 User information

The *User Info* block displays the name of the logged on user, an option that lets the user edit his or her personal information and the *Log off* option. The *Edit User Info* is optional and depends on the users role.

In the default the user User Info block is dark blue and placed just above the Menu bar

In the default skin the *User Info* bar is placed just above the *Menu* bar. Using styling parameters the two seem to be surrounded by border. This is achieved by adding a border to the top, left and right of the *User Info* bar and a border to the bottom, left and right of the *Menu* bar.

4.4.1 CSS Lists

The *User Info* bar is created using an HTML list. This allows you to easily change its orientation from horizontal to vertical using CSS styling. Below the CSS code for the user info items in the *style-element-userinfo.php* file.

```
#userinfo ul
{
  padding-right: 5px;
  float: right;
}


#userinfo li
{
  display: inline;
  border-right : 1px solid silver;
  margin: 0;
  padding: 2px 0.4em;
  font-size:10px;
  font-size-adjust:none;
  line-height: 2.2em;
}

#userinfo li a
{
  text-decoration: none;
}

#userinfo li a:hover
{
  text-decoration: underline;
}
```

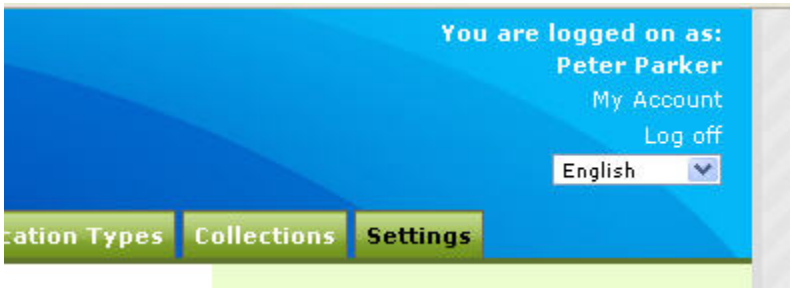
By removing the *display: inline* attribute of the <li> tag the orientation is changed from horizontal to vertical.

4.4.2 Adding additional information

The *User Info* bar has additional information. The code in the template file uses *generateUserInfo* to retrieve the *cAdditionalInfo* text from the language strings. The value for this string can be changed using the *Edit Language* page in the *Settings* section. This variable is used for a global language dependent message. In the default setup the value for this string is *PSM Web*. Hard coded text and other elements can be added to the HTML code.

```
<? if(function_exists("generateUserInfo")) { ?>
 <div id="userinfo">
  <div>
    <h3><? generateAdditionalInfo(); ?></h3>
    <? generateUserInfo(); ?>
  </div>
 </div>
<? } ?>
```

Below you'll see a variant of the *User Info* bar. In this case the orientation is placed vertical and a custom string was added.



A variant of the User Info bar showing additional information.

4.4.3 Highlighting "hovered" items

The *hover* attribute set for the *a* tag will render a different background color when the user places the cursor over this item. Additional parameters can be added to change the style of the text (for example the text color, font weight, text decoration or font style).

4.5 Menu bar

The *Menu* bar provides access to the main sections of PSM Web. Like the *User Info* bar the items of the *Menu* bar are created using HTML lists, which means that you can change its orientation by altering the CSS file.

In the default skin the Menu bar is placed directly beneath the User Info bar

4.5.1 Styling the menu items

The *Menu* bar is generated using HTML lists, within the list items hyperlinks (<a> tags) are used to define the target when the user clicks the item. The code snippet below shows a HTML code generated for a menu bar.

```
<div id="menu">
  <div>
    <ul>
      <li id="menuitem_1" class="selected">
        <a href="/site.php?pageid=welcome&amp;page=1&amp;init=true">Home</a>
      </li>
      <li id="menuitem_2">
        <a href="/site.php?pageid=storefront_overview&amp;page=1&amp;init=true">
          New Document
        </a>
      </li>
      <li id="menuitem_3">
        <a href="/site.php?pageid=ordermanager_overview&amp;page=1&amp;init=true">
          Order Manager
        </a>
      </li>
    </ul>
  </div>
</div>
```

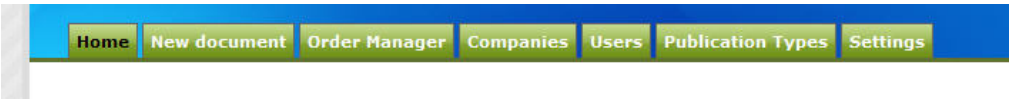
Below you'll the see CSS of the menu items in *style-element-menu.php* file. The following CSS code is used to show a horizontal menu bar. By removing the *display: inline* attribute of the <li> tag the orientation is changed from horizontal to vertical.

```
/* Menu */
/*
  Remove the commented lines to use the web design contrast color in the menu.
  Comment the background lines to hide the gradient image
*/

#menu
{
  border-width: 0 1px 1px 1px;
  border-color: #CDCDCD;
  border-style: solid;
  width: 100%;
  /* background-color: <? generateSkinContrastColorHighLight(1); ?>; */
  background: #D9E1E5 url('images/nav_bg.gif');
  height: 26px
}
```

```
#menu a
{
  /* background-color: <? generateSkinContrastColorHighLight(1); ?>; */
  background: #D9E1E5 url('images/nav_bg.gif');

  border-right: 1px solid #AFBEC7;
  color: #456;
  display: block;
  float: left;
  text-decoration: none;
  font-weight: bold;
  font-family: sans-serif;
  font-size: 1em;
  line-height: 25px;
  padding: 0 10px;
}

#menu li.selected a
{
  /* background-color: <? generateSkinContrastColorHighLight(1.3); ?>; */
  background-position: left bottom;
  color: #234;
}

#menu a:hover
{
  /* background-color: <? generateSkinContrastColorHighLight(0.9); ?>; */
  background-position: left bottom;
  color: #234;
}

#menu ul
{
  display: inline;
  list-style: none;
}

#menu li
{
  float: left;
  margin:0;
  padding:0
}
```

4.5.2 Creating a Tabbed menu

Below is a sample of the *Menu* bar where the menu items have a tabbed style. To achieve this a repeating background image is added to the <li> tag. Space is added between the items by adding a right margin. To complete the design border attributes are set for top, left and right sides of the <li> tag.



The Menu bar with repeated background image

```
#menu li {
  display: inline;
  border-width: 1px 1px 0 1px;
  border-color: #666666;
  border-style: solid;
  padding: 5px;  1
  line-height: 2em;
  margin-right: 3px;
  margin-left: 0;
  background-image: url("images/btn-bg.gif");
  background-repeat: repeat-x;
}
```

4.5.3 Styling the selected menu item

PSM Web will add an additional class to the selected menu item. This allows the style for the selected menu item to change to make it stand out. The color, font-weight, background color or background image can be used to create various effects.

```
#menu li.selected a {
 color: black;
}
```

The following code snippet shows how both the font color and background color of the selected item can be changed.

```
#menu li.selected a {
 color: black;
 background-color: <? generateSkinContrastColorHighLight(1.1); ?>;
}
```

4.6 Background

The <body> tag has two attributes to specify backgrounds. The background can be a color and/or an image. These attributes can be added directly in the *Template.php* file (bgcolor and background) or to the *Style.php* file.

4.6.1 Adding a background image

Background-images can be used in most HTML elements - not just for the whole page (body) and can be used for simple but effective results, such as rounded corners. Example:

```
body {
 background-image: url("images/main-bg.gif");
}
```

4.7 Overview tables

Overview pages show tabular data (for example the Companies, Users and Publication Types pages). An overview table is build using the table tag and follows the regular HTML <table> tag structure.

The Companies overview table

An overview table consists of the following elements:

- Head (dark blue bar)
- Subhead
- Content (the actual records)
- Footer (navigation/browse bar)
- Buttons (tool bar)

Looking at the source of an overview table page you can see that the table is part of a more complex hierarchy. The following code snippet shows a stripped version of an overview page:

```
<div class="content" id="companyOverviewFormContent">
  <table cellspacing="0" cellpadding="0" id="tableOverview" class="cCompanyOverview">
    <thead>
      <tr>
        <th colspan="8" id="cCompanyOverviewHeader">
          Companies overview<span class="recordsfound">(3 items)</span>
       </th>
      </tr>
    </thead>
    <tbody>
      <tr class="subhead">
        <td class="checkbox" id="select_header">
          <input type="checkbox" value="0" name="select" id="headerField"
          class="formCheckbox " onclick="doSelectAll()"/>
        </td>
        <td class="icon" id="fbCheckLogin_header">
          <span class="icon_online_status"/>
        </td>
        <td colspan="2" class="static" id="cName_header">Name</td>
        <td class="static" id="cCompanyCode_header">Company code</td>
        <td class="icon" id="_header"><span class="icon_pubtypes"/></td>
        <td class="icon" id="_header"><span class="icon_group"/></td>
        <td class="icon" id="_header"><span class="icon_user"/></td>
      </tr>
      <tr class="record level0">
        <td class="checkbox" id="fnCheckCompanyID[]_2">
          <input type="checkbox" value="2" name="fnCheckCompanyID[]"
          id="2Field" class="formCheckbox "/>
        </td>
        ...
      </tr>
    </tbody>
```

```
  </table>
</div>
```

The content is enclosed by a element. Each page contains a <form> tag, which is used to catch information entered by the user or to perform an action when the user clicks a button.

The <form> tag has two main components: the actual table and the toolbar/buttons to perform specific actions. A page will only contain a single overview table.

4.7.1 Head

The <th> tag indicates that the table cell(s) of the first row is a header cell, in this case the title for the table. In the *Style.php* file an id selector with a subselector of <th> is created to avoid styling conflicts with other tables.

```
#tableOverview th {
 background-color: <? generateSkinMainColor(); ?>;
 padding: 2px;
 font-weight: bold;
 text-align: left;
 border-width: 0 0 1px 0;
 border-color: #FFFFFF #FFFFFF #FFFFFF #FFFFFF;
 border-style: solid;
 color: #FFFFFF
}
```

4.7.2 Subhead

The second row shows the names for the columns. Often the first column contains a check box to select and deselect (toggle) the records of the current page in the overview.

4.7.3 Content

The rows (<tr> tag) that form the body of the table have the *record* class selector. In hierarchical overviews a second class selector is used to specify the level in the hierarchy. The <td> tags contained by the table row can have different class selectors. This depends on the type of content, examples are: text, icon and check box.

In the skins supplied with PSW the background color for the element is set when the user places the cursor over a specific row (hovers).

```
#tableOverview tr.level0:hover,
#tableOverview tr.level1:hover,
#tableOverview tr.record:hover {
 background-color: #E6E6FF;
}
```

4.7.4 Footer

The last table row is used to display navigation options. The navigation options become visible when the overview contains more than the number of records set in the *setRowsPerPage* variable (default 15). Please refer to Special variables chapter for more information about this function.

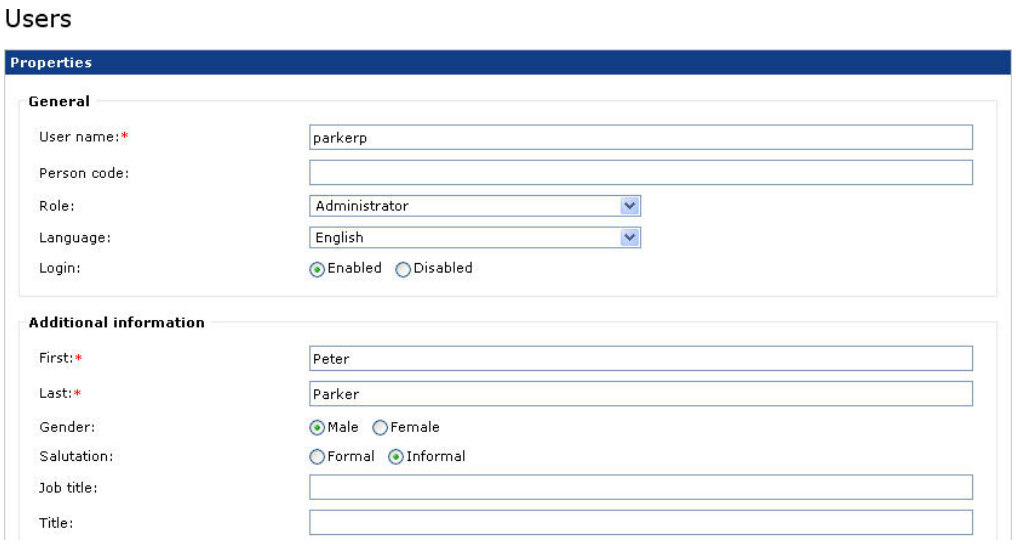
```
#tableOverview tr.level0:hover,
#tableOverview tr.level1:hover,
#tableOverview tr.record:hover {
 background-color: #E6E6FF;
}
```

4.7.5 Buttons

The buttons below the overview table are created using the <input> tag with a class *formButton*. The buttons have a unique id allowing you the specify the style per button. The buttons are enclosed by a tag with a class of *toolbar*.

4.8 Edit forms

A form is an area that can contain form fields. Form fields are objects that allow the visitor to enter information - for example text boxes, drop-down menus or radio buttons. When the visitor clicks a submit or save button, the content of the form is sent to PSM Web.



The Edit User Info form

The following sample shows a stripped down source code of an edit form. Like the overview pages all elements are surrounded by a content tag. The first child of this tag is the <form> tag.

```
<div id="content">
  <form action="site.php?formid=person_edit_form&amp;id=2"
  autocomplete="off" name="personEditForm" method="POST" id="personEditForm">

    <div class="content" id="personEditFormContent">

      <div id="cPropertiesHeader" class="formHeader">Properties</div>

      <fieldset class="formFieldset " id="cGeneralFieldset">
        <legend id="cGeneralLegend" class="formLegend">
          <span id="cGeneralLegendLabel" class="formLegendLabel">General</span>
        </legend>

        <div class="content">
          <div id="fcUserNameLine" class="formLine ">
            <span id="fcUserNameLabel" class="formLabel">
              User name:<span class="asterisk">*</span>
            </span>
            <span class="formElement">
              <input type="text" value="pp" name="fcUserName"
              id="fcUserNameField" class="formField"/>
            </span>
          </div>

          <div id="fcPersonCodeLine" class="formLine ">
            <span id="fcPersonCodeLabel" class="formLabel">Person code:</span>
            <span class="formElement">
              <input type="text" value="" name="fcPersonCode"
```

```
              id="fcPersonCodeField" class="formField"/>
            </span>
          </div>
        </div>

      </fieldset>

    </div>
  </form>
</div>
```

The <form> tag is used to process the information entered by the user. The contents of the <form> tag are captured by a with a unique id.

4.8.1 Form head

The first content element is the form header. In the default skin the information in this tag is used to visually group the elements in the form.

```
.formHeader {
  display: block;
  width: 100%;
  background-color: <? generateSkinMainColor(); ?>;
  color: #FFFFFF;
  padding: 2px;
  font-weight: bold;
}
```

4.8.2 Form lines

Each line in an edit form is enclosed by a tag. This element has an unique id and a *Formline* class. Normally each line contains two child elements: a label and a field. The form fields can be of various types, for example: plain text input, a pull down menu, radio buttons or a range of check boxes. The following snippet shows a form line containing a form label and a regular text input field. As the field requires input an additional tag is added for the asterisk symbol.

```
<div id="fcUserNameLine" class="formLine ">
  <span id="fcUserNameLabel" class="formLabel">
    User name:<span class="asterisk">*</span>
  </span>
  <span class="formElement">
    <input type="text" value="" name="fcUserName" id="fcUserNameField" class="formField"/>
  </span>
</div>
```

4.8.3 Warnings and Errors

Warnings and errors are hidden elements for which the visibility is changed when the actual warning/error occurs. A warning line has two child element: a warning label and a container element for the warning icon and message.

```
<div id="personEditInfoForm_fcPassword_Required" class="warnings">
  <div class="warningTable">
    <span class="warningLabel"> </span>
    <span class="warningIcon">
      <span class="warningItem">Password is a required field!</span>
    </span>
  </div>
</div>
```

The *warningLabel* is used as spacer to make sure that the warning icon and message align with the field of the accompanying form line. The *warningIcon* holds the actual warning message which is stored in a separate tag. The warning icon is set using CSS. The following snippet shows *background-image* attributes for the warning icon.

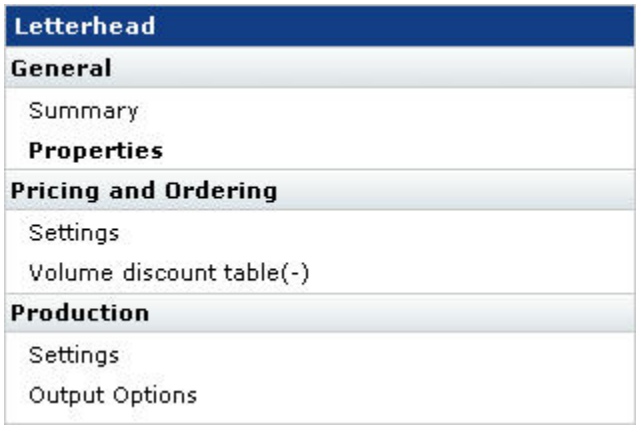
```
.warningIcon {
  display: inline-block;
  width: 73%;
  background-image: url(../../images/icon_alert.gif);
  background-repeat: no-repeat;
  background-position: 0px 3px;
  display: inline-block;
  vertical-align: top;
}
```

In this case the icon is coming from the image folder stored in the PSM Web web site root. You can use you own icon by placing an image within the skins folder (or an images folder in your skin folder) and changing the path to that image file.

A required field warning

4.9 Sub menus

Several sections have one or multiple sub menus. Sub menus are used to navigate to sub sections of that specific section. Sub menus have their own classes and styles.



The sub menu of the Settings section

Like the *Main* menu and the *User Info* block the sub menus are created using HTML lists (<ul> tag and <li> tags). The following code snippet shows source code of a sub menu (*Publication Type* section):

```
</div>
  <div id="submenu2">
   <div id="LetterheadID" class="content">
   <div class="formHeader">Letterhead</div>
   <span class="submenu">
     <ul>
      <li class="level0">General</li>
      <li class="level1">Summary</li>
      <li class="level1 selected">Properties</li>
      <li class="level0">Pricing and Ordering</li>
      <li class="level1">Settings</li>
      <li class="level1">Volume discount table (-)</li>
      <li class="level0">Production</li>
      ...
     </ul>
    </span>
  </div>
</div>
```

The list items have two class selectors. The first defines the indent level. The sub menu of the Settings section uses sub levels for the items that relate to the Email subsection. The second selector is optional and will only be set to mark the selected menu item. This allows you to visually mark the selected item by changing its style in the style documents.

```
#submenu li.level0{
  text-indent: 2px;
  background-color: <? generateSkinContrastColor(); ?>;
  background: #D9E1E5 url('images/nav_bg.gif');
  background-repeat: repeat-x;
  background-position: top left;
  border-top: 1px solid #CDCDCD;
  border-bottom: 1px solid #CDCDCD;
```

```
}

#submenu li.level1{
 padding-left: 1em;
}

#submenu .selected{
 font-weight: bold;
}
```

4.9.1 Multiple sub menus

The *Template.php* file allows pages to have multiple sub menus. The source code of such a page has a unique id for each sub menu. In the style documents these IDs are grouped by separating each selector with a comma:

```
#submenu li.level0, #submenu1 li.level0, #submenu2 li.level0{
}

#submenu li.level1, #submenu1 li.level1, #submenu2 li.level1{
 padding-left: 1.5em;
}

#submenu .selected, #submenu1 .selected, #submenu2 .selected {
 font-weight: bold;
}
```

5 Special variables

In this section we list the special variables you can use in the *Template.php* and *Style.php* files.

5.1 Variables for template files

The functions in this section can be used in templates files (e.g. *Template.php*).

5.1.1 generateString

The *generateString* function retrieves a language string based on the supplied parameter. It is used to insert language dependent text.

```
<? generateString(cSystemName); ?>

/* Renders to following text */
PrintShop Web
```

Custom strings can be added to the language strings. A string is made up of a name and a value, which are seperated by the = symbol.

```
cSystemName=PrintShop Mail Web
cAdditionalInfo=PSM Web
cFooter=PrintShop Mail Web, www.objectiflune.com
```

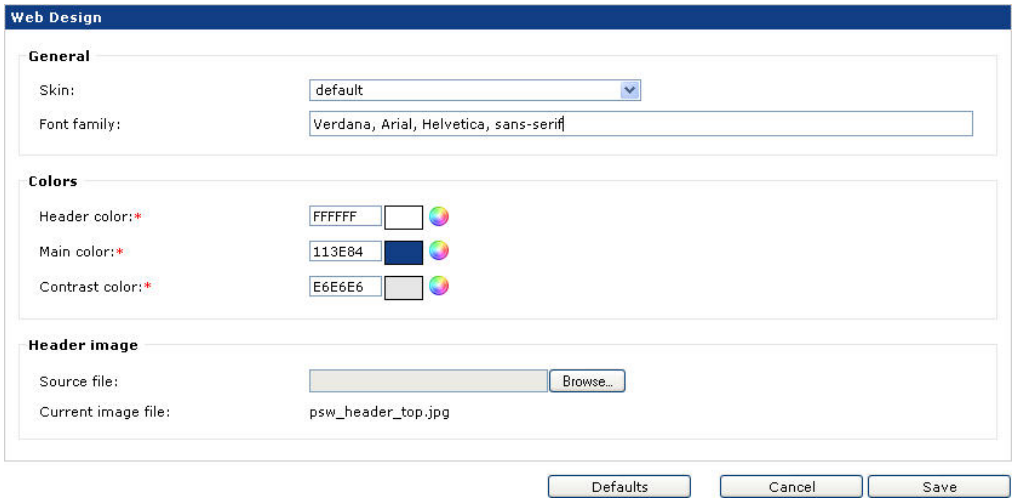
5.1.2 setRowsPerPage

The *setRowsPerPage* function can be used to change the number of rows shown in overview tables. The function can added to the *Template.php* file and should be insert before the *generateContent* function. By default overview tables show 15 rows per page.

```
<? setRowsPerPage(25); ?>

<div id="content">
  <? if (getFormCount() > 0){ ?>
    <? generateContent(); ?>
  <? } ?>
</div>
```

5.2 Variables for style files

The PSM Web style files (e.g. *Style.php*) are regular CSS files. Its a PHP file that renders a CSS file. By including the *skin_customization.php* line at the beginning of your *Style.php* file the values entered in the *Edit Web Design* pages can be used in your style. Once included special PHP functions can be used to retrieve style information that is entered in the *Edit Web Design* page of the companies in PSM Web.



The edit Web Design page

These values are retrieved from the PSM Web database and or calculated based on these values. You can set these values per company in the companies Web Design page. The first few lines of the *Style.php* document are as follows:

```
<?php include_once "../../functions/skin_customization.php" ?>

*
{
 margin:0;
 padding:0;
}

h1,h2,h3,h4,h5,h6,p,blockquote,form,label,ul,ol,dl,fieldset,address
{
 margin: 0;
}

li,dd
{
 margin-left:1em;
}
```

5.2.1 generateSkinContrastColor

The *generateSkinContrastColor* function retrieves the contrast color information set in the *Edit Web Design* page of the PSM Web site. In the default skin this color is used for the menu bar items, table subheads and table footers. Below an example on how this color can be set using this function:

```
#tableOverview .subhead td {
 background-color: <? generateSkinContrastColor(); ?>;
 border-width: 0 1px 0 0;
 border-color: #FFFFFF;
}
```

5.2.2 generateSkinContrastColorHighLight

The *generateSkinContrastColorHighLight* function retrieves the skins contrast color set in the *Edit Web Design* page. A parameter can be supplied to change the tint of the color. Fractional values are used, where 0.0 will be black, 1.0 the original color and higher values will result in a lighter tint of this color.

Below an example on how this color can be set using this function. In this case the background color of the selected menu item will have a lighter color:

```
#menu .selected a {
 background-color: <? generateSkinContrastColorHighLight(1.1); ?>;
}
```

5.2.3 generateSkinFont

The *generateSkinFont* function retrieves the font information set in the *Edit Web Design* page of the PSM Web site. The value entered in the *Edit Web Design* page (Font family) can be a single font name or a comma separated list. Below an example on how the *Font family* can be set using this function:

```
input, select, body, textarea {
 font-size: 70%;
 line-height: 1.3em;
 font-family: <? generateSkinFont(); ?>;
}
```

5.2.4 generateSkinHeaderColor

The *generateSkinHeaderColor* function retrieves the header color information set in the *Edit Web Design* page of the PSM Web site. The value entered in the *Edit Web Design* page can be entered manually or selected in the color picker. Typically this color will match the background color of the company logo. This color can be used to set the background so the logo or background image nicely blends in wich the overall design.

Below an example on how this color can be set using this function:

```
#header {
 height: 90px;
 background-image: url(<? generateSkinLogo(); ?>);
 background-position: center left;
 background-repeat: no-repeat;
 background-color: <? generateSkinHeaderColor();?>;
 width: 100%;
}
```

5.2.5 generateSkinHeaderColorHighLight

The *generateSkinHeaderColorHighLight* function retrieves the skins header color set in the *Edit Web Design* page. A parameter llows the tint of the color to be changed. Fractional values are used, where 0.0 will be black, 1.0 the original color and higher values will result in a lighter tint of this color.

Below an example on how this color can be set using this function:

```
#tableOverview td {
 padding: 2px;
 border-width: 0 0 1px 0;
 border-color: background-color: <? generateSkinHeaderColorHighLight(1.1); ?>;
 border-style: solid;
}
```

5.2.6 generateSkinLocation

This functions returns the path to your skin folder. It can be used in both style files and the template file to insert the path to the folder of the skin. This prevents you from entering the full path making it easier to duplicate a skin folder.

```
<img src="<? echo generateSkinLocation() ?>/banner-image1.png"/>
```

5.2.7 generateSkinLogo

The *generateSkinLogo* retrieves the path to the header image specified in the *Edit Web Design* page. If no image is specified the path of the header image from the system default skin will be returned. Below an example on how this function can be used in your *Style.php* file:

```
#header {
 height: 90px;
 background-image: url(<? generateSkinLogo(); ?>);
 background-position: center left;
 background-repeat: no-repeat;
 background-color: <? generateSkinHeaderColor();?>;
 width: 100%;
}
```

5.2.8 generateSkinMainColor

The *generateSkinMainColor* function retrieves the main color set in the *Edit Web Design* page of the PSM Web site. The value entered in the *Edit Web Design* page can be entered manually or selected in the color picker. Below an example on how this color can be set using this function. In this case the background color is set for the edit forms header bar class (or title bar).

```
.formHeader {
 display: block;
 width: 100%;
 background-color: <? generateSkinMainColor(); ?>;
 color: #FFFFFF;
 padding: 2px;
 font-weight: bold;
}
```

5.2.9 generateSkinMainColorHighLight

The *generateSkinMainColorHighLight* function retrieves the skins main color set in the *Edit Web Design* page. The parameter of this functions lets you change the tint of the color. Fractional values are used, where 0.0 will be black, 1.0 the original color and higher values will result in a lighter tint of the main color.

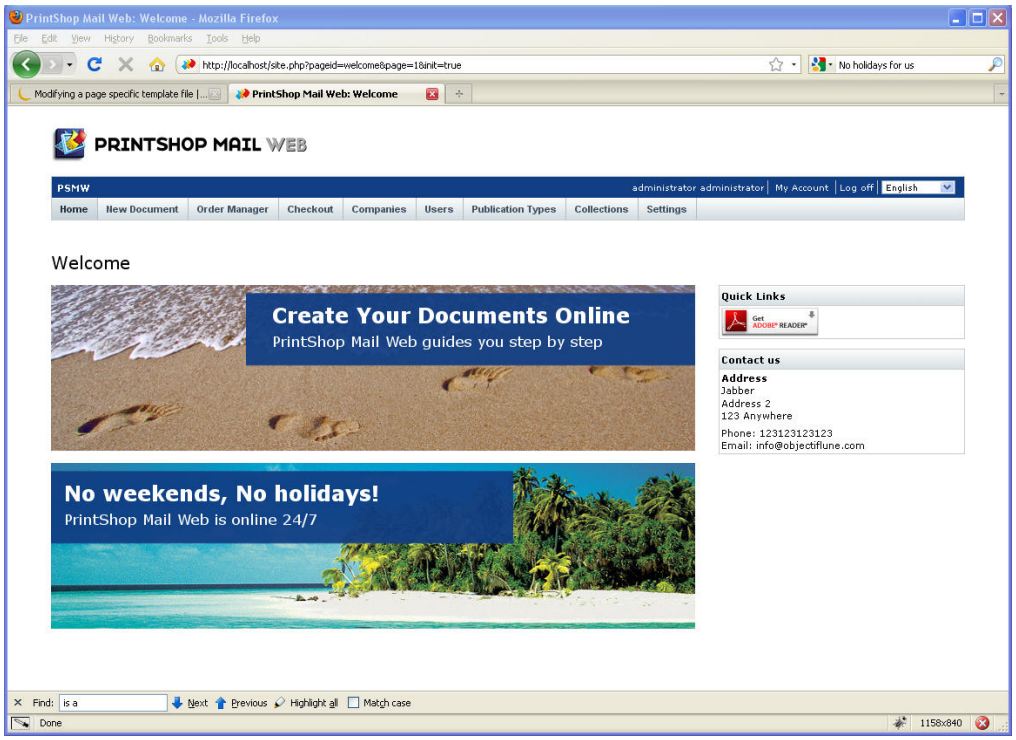
Below an example on how this color can be set using this function. In this case the background color of the user info bar will have a darker tint when the user places the cursor over these items (hover).

```
#userinfo li a:hover{
 background-color: <? generateSkinMainColorHighLight(0.6); ?>;
 line-height: 2em;
}
```

6 Creating page exceptions

The Template.php file contains the main outline of the PSM Web web site, it contains HTM code with a few snippets of PHP. It is invoked for each page request and outputs the final page contents.

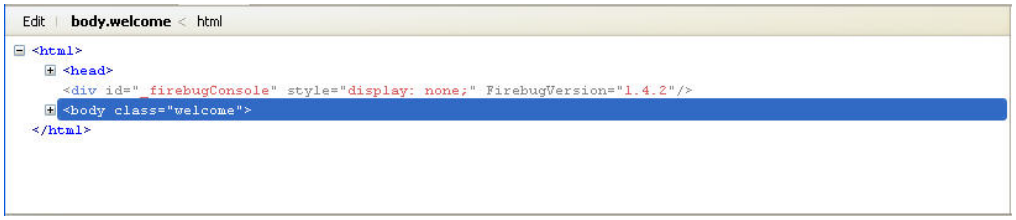
The skinning engine lets you create templates for pages (page exceptions). This method can be used to add information and/or custom functionalities to a specific page.



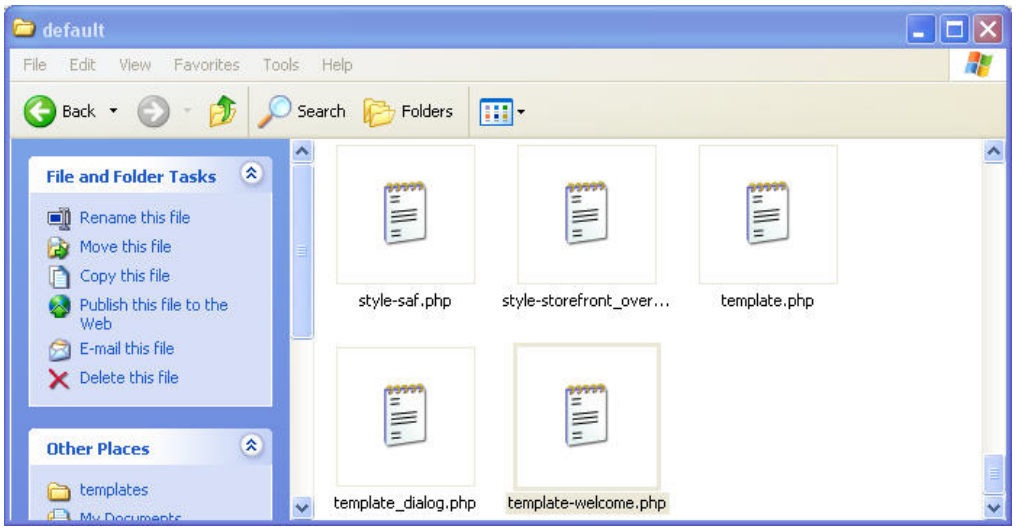
The Welcome page showing a banner image and custom side menus.

6.1 Creating a page specific template file

To create a page specific template file you should create a copy of the main *template.php* file in the directory of your skin. Then, you need to rename to the file to template-<pageid>.php. The ID of the page should match the name of the class name of the body-element of that page (view the HTML source of the page in your browser or use webdevelopment plug-in like Firebug for Fire Fox).



Use the class name of the body element in the template name (<pageid>).



A template exception file for the Welcome page.

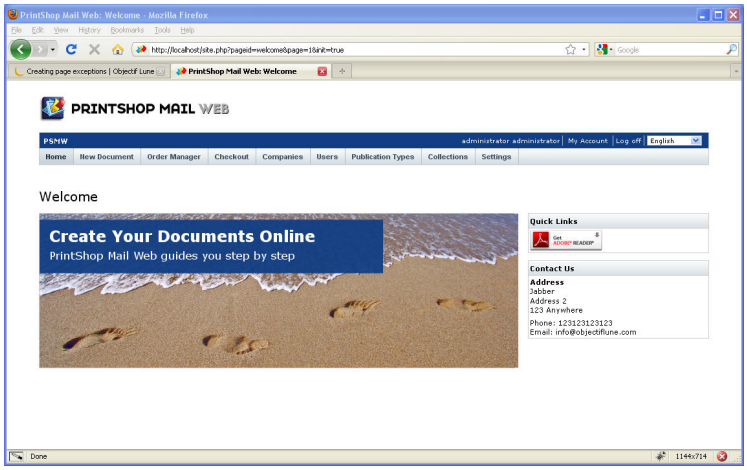
6.2 Modifying a page specific template file

The duplicated template file contains various function blocks that render the final output. The position of these blocks combined with the style files define the presentation of the web site. By inserting HTML and PHP code you can add functions and information to the page. In some cases you might want to *remove* unwanted elements from the template, be very *careful* when experimenting with this at it might lead to a non working skin.

The following HTML/PHP code was used to replace the text and sidemenu elements of the Welcome page. The content selection is replaced by an banner image and side menus are added for some quick links and contact information. Note that in this sample we added style information in-line, typically this information is added to one of the style files (or a custom style file). The images used in this sample are stored in the folder of the skin.

```
<div id="content">
  <div style="float:left;width:100%;">
    <ul style="list-style-type: none;margin:0;" id="banners">
      <li style="margin:0 0 1em 0;">
        <img src="<? echo generateSkinLocation() ?>/banner-steps.png"/>
      </li>
      <li style="margin:0 0 1em 0;">
        <img src="<? echo generateSkinLocation() ?>/banner-holiday.png"/>
      </li>
    </ul>
  </div>
</div>

<div style="clear:right;float:right;width: 27%;">
  <div style="border:1px solid #CDCDCD;">
    <p style="
      background:#D9E1E5 url(<? echo generateSkinLocation() ?>/images/nav_bg.gif)
      repeat-x scroll left top;
      border-bottom:1px solid #CDCDCD;line-height:2em;
      text-indent:2px;font-weight:bold;">Quick Links</p>
    <div style="padding: 2px">
      <p><img src="images/get_adobe_reader.gif"/></p>
    </div>
  </div>
  <div style="border:1px solid #CDCDCD;margin-top: 1em;">
    <p style="background:#D9E1E5 url(<? echo generateSkinLocation() ?>/images/nav_bg.gif)
    repeat-x scroll left top;
    border-bottom:1px solid #CDCDCD;line-height:2em;
    text-indent:2px;font-weight:bold;">Contact us</p>
    <div style="padding: 2px">
      <p style="margin-bottom: 0.5em;">
        <b>Address</b></br>Jabber<br/>Address 2</br>123 Anywhere
      <p>
      <p>Phone: 123123123123<br/>Email: info@objectiflune.com</p>
    </div>
  </div>
</div>
```

The following image shows the outcome of the page specific template exception.



A custom welcome page.

As the web site is created in PHP custom statements could be added based (for example based on the logged on user). With a little help of Javascript (jQuery) you could create an image rotator to show images as slide show. The following code creates a very simple image rotator. Note that we added *display:none* to the second banner in order to initially hide the image.

```
<div id="content">
  <div style="float:left;width:100%;">
    <ul style="list-style-type: none;margin:0;" id="banners">
      <li style="margin:0 0 1em 0;">
        <img src="<? echo generateSkinLocation() ?>/banner-steps.png"/>
      </li>
      <li style="margin:0 0 1em 0;display:none;">
        <img src="<? echo generateSkinLocation() ?>/banner-holiday.png"/>
      </li>
    </ul>
  </div>
</div>

<script type="text/javascript">
  $(document).ready(function() {
    setInterval('rotateBanners()',5000);
  });

  function rotateBanners() {
    var currentBanner = $('#banners li:visible');
    var firstBanner = $('#banners li:first');

    //Get next banner, when it reaches the end, rotate it back to the first banner
    if(currentBanner.next().length){
      currentBanner.next().fadeIn(1000);
      currentBanner.hide();
    } else {
      firstBanner.fadeIn(1000);
      currentBanner.hide();
   }
  }
</script>
```

7 DOM manipulation using jQuery

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. With the introduction of PrintShop Mail Web 2.x the jQuery library is part of the PSM Web installation. Information about jQuery can be found at: http://www.jquery.com

The following sections describe the jQuery basics and give you helpful tips about how to use jQuery in PSM Web.

7.1 Launching code on Document Ready

jQuery has a simple statement that checks the `document` and waits until it's ready to be manipulated, known as the ready event. Inside the ready event you can add the code that you want to run right when the page is loaded.

```
<script type="text/javascript">
  $(document).ready( function(){
    //Your code here
  });
</script>
```

You can add this Javascript code in the head section of the template.php file or create an external Javascript file. If an external Javascript file is used make sure that it is added (inlcuded) in the head section of the template.php file. The following snippet shows how to include a Javascript file (*template-welcome.js*) located in the folder of the skin.

```
<html>
 <head>
   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title><? generateSystemName(); ?>: <? generateTitle(); ?></title>
   <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon">
   <? generateCSSIncludes(); ?>
   <? generateJavaScriptIncludes(false); ?>
   <script src="<? echo generateSkinLocation() ?>/template-welcome.js"></script>
 </head>
 <body class="<? generatePageID(); ?>">
   ...
 </body>
</html>
```

The following could be the content of the Javascript file.

```
$(document).ready(function() {
    alert('hello world');
});
```

7.2 Populating fields with computed values

The following sample populates a user input field of a PSM document with the current date (New Document section). First a string is created (*toDay*) with the current date. The last line in the code retrieves the input field based on the fields label (Today:) and populates the input field with the computed date string.

```
<script type="text/javascript">
  $(document).ready( function(){

    var currentDate = new Date()
    var month = currentDate.getMonth() + 1
    var day = currentDate.getDate()
    var year = currentDate.getFullYear()
    var toDay = month + "/" + day + "/" + year

    $("span:contains('Today:')").next("input").val(toDay);

  });
</script>
```

Todays date is automatically entered in the input field.

The code sample could be added directly into the template.php file and/or included using an external Javascript file.

7.3 Removing elements from the DOM

The following snippet is a simple sample on how to remove an HTML element based on its ID. The element is removed once the page is rendered by the browser.

```
<script type="text/javascript">
  $(document).ready( function(){
    $("#legend_joboverviewList").remove();
  });
</script>
```

Removing elements is not without risk and should be tested carefully (especially save actions).

7.4 Adding information to the DOM

The following snippet adds an HTML snippet to the DOM. An comment line is added to the web form of the User Input fields page.

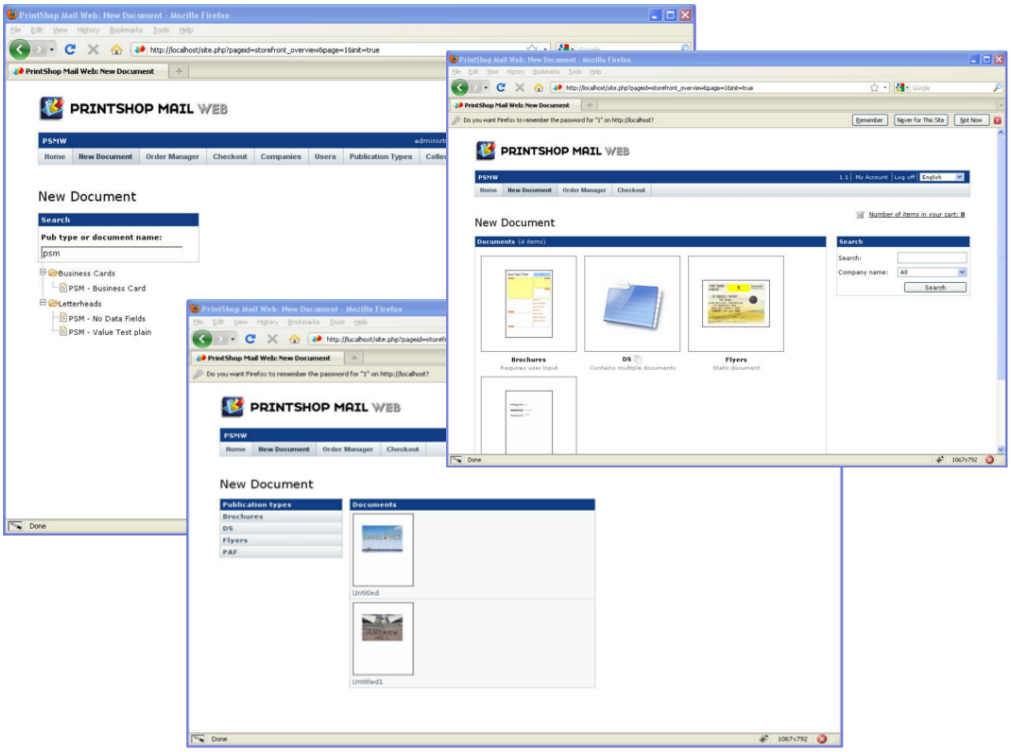
```
<script type="text/javascript">
  $(document).ready( function(){

    var  formline = "<div class=\"formLine\">" +
    "<span class=\"formLabel\" style=\"width: 100%\">" +
    "<i>All fields signed with a <span class=\"asterisk\">*" +
    "</span> are mandatory</i></span></div>";

    $("body.preview_userinput #content div.content").append(formline );

  });
</script>
```

8 Customizing the store front

Combining template page exceptions with skinning techniques allows the creation of custom store front pages. This can be used to create a unique design with for example a tree view or accordeon menu. To achieve this the skinning engine includes a special function to retrieve a list of documents and publication in a PHP array. This data can be used to generate custom HTML code in the template file (preferably a template exception for the store front pages).

The PHP getTree() function of the Storefront class retrieves a list of publication types and documents that the current user has access to.

Store front samples can be found in the folder of the *default* skin. They can be used as a reference when creating your own pages. The samples use template exception for the store front page and custom Javascript/ jQuert code.



8.1 Storefront class

The Storefront class lets web designers and developers retrieve publication type and document information to create custom store front pages. The following sections describe the functions of this class:

getTree();

getTree();
Description *getTree(string `$searchfor`, string `$delimiter`, int `$publicationtypeid`)*
Parameters • $searchfor: the string to search for (leave emtpy to return all publication types and templates). • $delimiter: the boundary string to create virutal sub folders. • $publicationtypeid: return the documents of the specified publicationtype id.
Return Values An associative array.

The following PHP code shows how to implement this getTree() function:

```
<?php
//Retrieve the publication types and templates for the logged on user
$storeFront = New Storefront();
$treeData = $storeFront->getTree();
?>
```

The following code is a sample of the array returned by the getTree() function.

```
[1] => Array
(
 [id] => 1
 [name] => Business Cards
 ...
 [documents] => Array
   (
   [1] => Array
     (
     [id] => 1
     [name] => BC-Portrait
      ...
     )
   [2] => Array
     (
     [id] => 2
     [name] => BC-Landscape
     ...
     )
   )
)
```

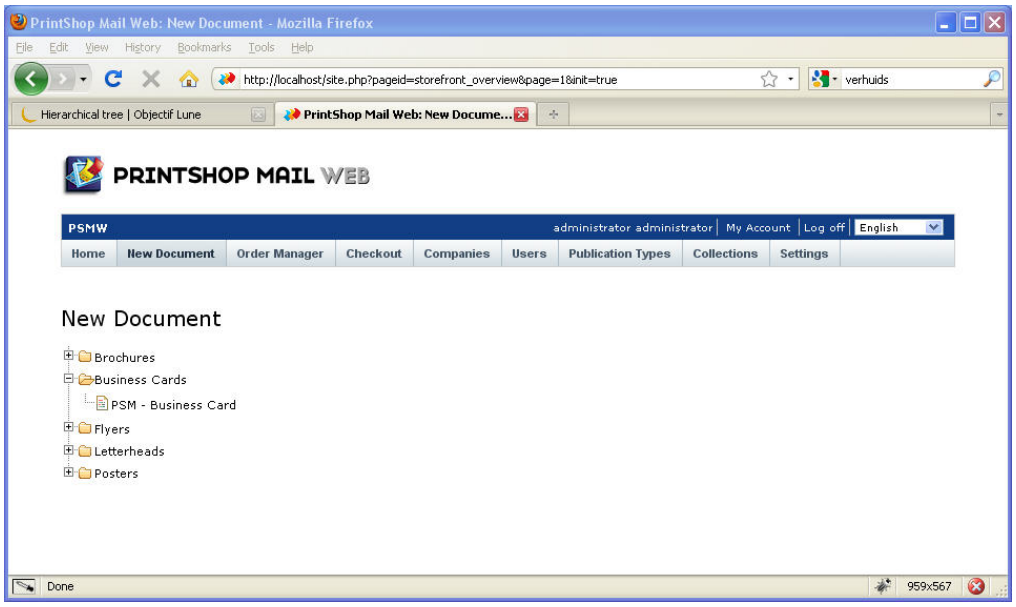
The snippet below shows how to use this information in a link (<a>). In order to show the correct subsequent page in the ordering workflow the link should refer the *preview_init_form* page (fromid parameter of the url).

```
<a href='site.php?formid=preview_init_form&id=<?php echo $document['id']; ?>'>
<?php echo $document['name']; ?>
</a>
```

8.2 Creating a hierarchical tree

This example explains how to replace the default thumbnail view of the store front with a hierachical tree view. The example combines PHP, CSS and makes use of the Treeview plugin for jQuery to hide and show branches of a tree.

To install the example copy the *template-storefront_overview.php* from the *storefront_tree_template* folder to the root of the *default* skin.



The store front using a hierarchical view (tree view).

The sample file contains links to external Javascript files and some custom code to generate the actual tree.

The Javascript file

This is the Javascript code stored in the *feil "tree.js":*

```
$(document).ready( function(){
  $("ul.filetree").treeview({
    collapsed: true
});
```

The code invokes the Treeview plugin once the browser has finished loading the page. The Treeview pluging uses a standard HTML <ul> element (unordered list) to render the tree.

The template file

The actual code that retrieves the publication type and document list is placed in the contect block of the template file (template-storefront_overview.php). The first section instantiates the *Storefront*. The getTree() returns a hierachical array of the publication types and documents. In the following section we will access each *array* item using the *foreach* function of PHP.

```
//Retrieve the publication types and templates for the logged on user
$storeFront = New Storefront();
$treeData = $storeFront->getTree();
```

Once the publication type and document information is collected in a PHP array we need to iterate through the array values. For this we use a PHP foreach loop. Within this function a custom PHP function (*renderPublicationType*) is called to generate the actual HTML.

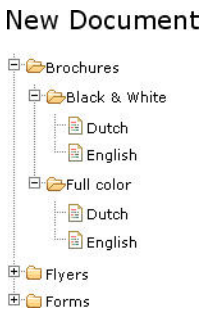
This function generates the <li> elements required by the Treeview plugin. When a publication type contains one or multiple documents a sub list is created, again using <ul> and <li> elements. For each document a link to the *preview_init_form* is inserted supplying the internal ID of the document. The functions of the *preview_init_form* file determine the workflow of that document (e.g. show the User Input fields page or Database Upload page).

```
foreach ($treeDataArray as $publicationTypes)
{
  renderPublicationType($publicationTypes);
}

function renderPublicationType($publicationType)
{
  echo "<li><span class='folder'>" . $publicationType['name'] . "</span><ul>";

  if (isset($publicationType['publicationTypes'])) {
    foreach ($publicationType['publicationTypes'] as $subPublicationType) {
      renderPublicationType($subPublicationType);
    }
  }

  if (isset($publicationType['documents'])) {
    foreach ($publicationType['documents'] as $document) {
      echo "<li><span class='file'>";
      echo "<a href='site.php?formid=preview_init_form&id=" . $document['id'] . "'>";
      echo $document['name'];
      echo "</a>";
      echo "</span></li>";
    }
  } else {
    if (!isset($publicationType['publicationTypes'])) {
      echo "<li><a href='#'>-</a></li>";
    }
  }

  echo "</ul></li>";
}
```

Virtual Subfolders

In the current version of PSM Web a publication type can not contain subfolders. You can however simulate nested publication types. The getTree() function can create virtual sub publication types or sub folders based on a delimiter character that appears in a document name.

When you call the function as follows getTree('', '_');, it will split the template names using the *underscore* character (if it exists in the name). The parts are used for the virtual folder name, the last part is used as the document name.



The tree view with virtual subfolders.

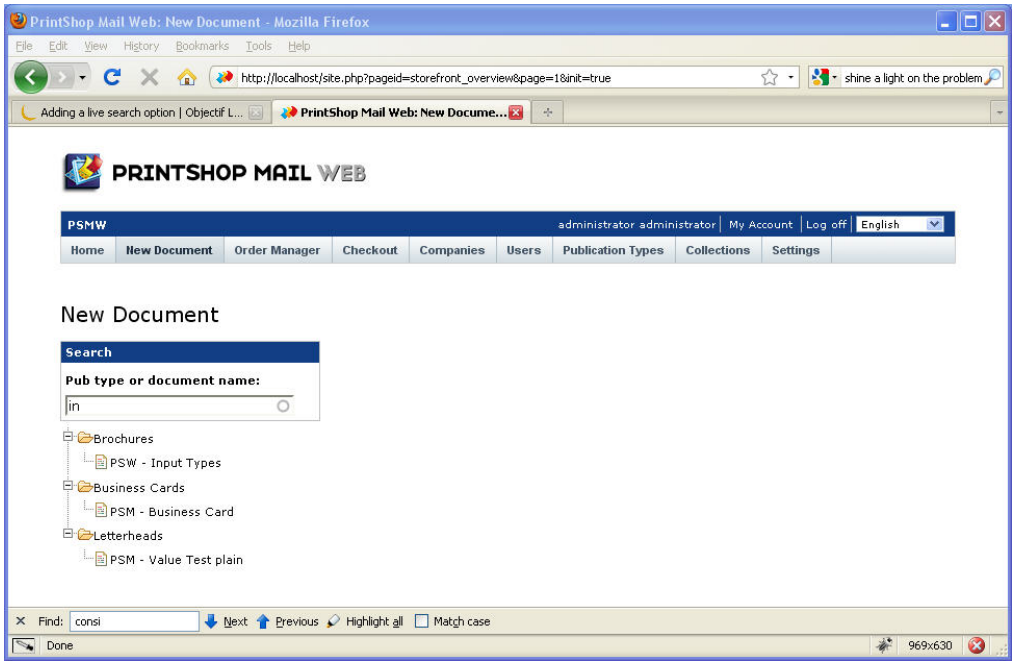
The following shows the returned array:

```
Array (
  [1] => Array  (
    [id] => 1
    [name] => Brochures
    ...
    [publicationTypes] => Array
    (
      [Black & White] => Array (
        [name] => Black & White
        [documents] => Array (
            [1] => Array (
              [id] => 1
              [name] => Dutch
              ...
            )
            [2] => Array (
              [id] => 2
              [name] => English
              ...
            )
         )
      )
      [Full color] => Array (
        [name] => Full color
        [documents] => Array (
            [1] => Array (
              [id] => 1
              [name] => Dutch
              ...
            )
            [2] => Array (
              [id] => 2
              [name] => English
              ...
            )
         )
      )
    )
  )
)
```

8.3 Adding a live search option

This example demonstrates a live search, where you get search results while you type. An event is triggered when the user presses, and releases a key in the input field. When the event is triggered, a Javacscript function is executed. The result of this function is used to update a placeholder element on the web page. The function performs an AJAX request to retrieve the document list from the database. AJAX (Asynchronous JavaScript and XML) is a method of building interactive applications for the Web that process user requests immediately.

To install the sample copy the *template-storefront_overview.php* from the *storefront_tree_livesearch* folder to the root of the *default* skin.



The tree view with a search block.

This sample constists of the following parts:

- Store front specific template file (exception file). The template file contains links to external JavaScript files and some custom HTML code for the search and tree areas.
- An php file that retrieves the document list. The result set is used to generate the HTML for the tree. This file is called by a custom Javascript function in the <head> section of the template file. This functions calls this php file using an AJAX request.

The Javascript file

This is the JavaScript code stored in the file "livesearch.js":

```
$(document).ready( function(){
    getTree();
    generateTree();
});

function getTree(){
    var searchfor = $('#searchfor').val();
    $('#searchfor').css({backgroundPosition: '100% -18px'})

    $.ajax({
        type: "GET",
        url: "templates/default/storefront_tree_livesearch/livesearch.php",
        data: "searchfor=" + searchfor,
        success: function(msg){
            $('#searchfor').css({backgroundPosition: '100% 2px'})
            $("#tree").html(msg);
            generateTree();
        }
    });
}

function generateTree(){
    $("#tree ul").treeview({
        collapsed: true
    });
}
```

$(document).ready

This code is executed when the browser is finished loading the web page. It executes the *getTree()* and *generateTree()* function. This section makes sure that the full list is retrieved when the store front page is invoked (New Document).

getTree()

This function executes every time a character is entered in the input field. It retrieves the text from the search field and sends the contents to the *livesearch.php* file on the server. The HTML code returned by the *livesearch.php* file is used to update a placeholder element on the page ( ).

generateTree()

Once the tree information is loaded in to the DOM the generateTree() function is called to apply the final tree view and tree functionality.

The template file

In the content area of the template file (template-storefront_overview.php) an input field is added. The *getTree()* function is added to the onkeyup event of the <input> field. This event is triggered when the user presses, and releases a key in the input field.

```
<div id="content">
  <div class="search" style="border:1px solid silver;width:250px;margin-bottom:0.5em;">
    <div id="cSearchHeader" class="formHeader">Search</div>
    <div style="padding: 4px;">
      <p><b>Pub type or document name:</b></p>
      <input style="
        background-image:url('<? generateSkinLocation(); ?>/
        storefront_tree_search/images_tree/throbber.gif');
        background-repeat:no-repeat;background-position:100% 2px;
        width:220px;margin-top:0.5em;" id="searchfor" onkeyup="getTree(this.value)"/>
    </div>
  </div>
  <div id="tree"></div>
</div>
```

Livesearch.php

The PHP page called by the JavaScript code is called "livesearch.php". The code retrieves the publication types and templates on the server based by the search string entered by the visitor.

The getTree() accepts a search string parameter (getTree('cards')). The getTree() function returns a list of publication types and documents where the case sensitive search string appears at least in one of the fields:

- Document name
- Document code
- Publication type name
- Publication type code

The search result is used to create an hierchical tree using unorderedlists (<ul>). The response is used by the getList() function