

Templates Creation

This document gives the guidelines on how to work with templates to ensure a problem-free operation of SpinPike CMS. The example of creating a template is given.

Introduction.....	3
Constants.....	4
Variables.....	4
\$Site.....	4
\$Section.....	4
\$Item.....	5
\$Cart.....	5
Objects.....	5
Module.....	5
Language.....	6
Version.....	6
Template.....	6
User_Group.....	7
User.....	7
Administrator.....	7
Content.....	7
Content Item.....	8
Image.....	8
File.....	9
Log.....	9
"Item" object / Item.....	9
"Shopping cart" object / Cart.....	9
"Online payment systems" object / Paysystem.....	9
"Order" object / Order.....	10
Template functions.....	10
Site versions functions.....	10
a_version.....	10
a_versions.....	11
Site sections functions.....	11
a_section.....	11
a_sections.....	11
a_section_trail.....	12
a_sitemap.....	12
a_sections_tab.....	12
a_sections_level.....	13
a_sections_repmenu.....	13
a_sections_count.....	13
a_sections_rand.....	13
Site content module functions.....	14
a_section_content.....	14
a_section_content_items.....	14
a_eshop_items.....	14
a_eshop_show_info.....	15
a_eshop_offer.....	15
a_eshop_add2cart.....	15
a_eshop_add2compare.....	16
a_eshop_compare.....	16
a_eshop_paysystems.....	16
Integrating modules.....	17
"Product list" template.....	17
"Product card" template.....	18
"Shopping cart" template.....	19
"Product comparison" template.....	21
"Buyer details" form; "Thank you" page.....	23
Template creation example.....	23
Entering the site name and META-information.....	25
Creating the main menu.....	25
Outputting news headlines.....	26
Inserting a replicated navigation.....	26

If you have any questions that have not been covered in the given document, please contact our technical support service at support@savvybox.com.

Introduction

The templates define the pages layout of an SpinPike powered site. A template is an HTML page with incorporated dynamic blocks in it. These blocks are realized using Smarty (<http://smarty.php.net>). The Smarty-related documentation is available at <http://smarty.php.net/docs.php>.

The template files have a `tmpl` extension. The template creation process goes as follows:

1. When SpinPike is installed some of the templates are generated automatically. These templates provide for the system correct operation.
Besides, a special template directory tree is generated to store the templates. For example, the Content module is responsible for managing the site content. Correspondingly, a Content directory is created that stores a template for the site homepage named `index.tmpl`.
2. You can modify templates either through the administration interface or on the server using any available code editor (for example, HomeSite or Dreamweaver).
3. To add a template when in the administration area, you need to browse through Modules -> Templates, select the module to which you want to add a template and press Add Template. Enter the name for the template (for example, Homepage), enter the file name and press Apply.

After that the template can be modified.

The given document gives a detailed description of the template creation process. In brief, the process goes like this:

1. A static HTML template is created.
2. The template is either put on the server or its code is copied and pasted through the SpinPike™ administration interface.
3. Finally, the dynamic blocks such as menu or content, etc., are embedded in the templates.

A template can contain pictures and have style sheets and JavaScript files linked to it. Because of the Smarty peculiarities, you cannot use such symbols as “{”, “}” in templates. It is recommended that you link style sheets and JavaScript files externally, and not embed them in the template. However, you may use special command `{literal}` `{/literal}` for escaping { and } symbols.

Pictures and all external files are uploaded to the root directory. For instance, if the server root directory is named `html`, and the pictures are stored in the `images` folder, you need to upload the `images` folder into the `html` directory. As a result, the folders will be arranged as follows: `html/images/`. That is you follow the same procedure as if you were developing a common static site.

Below are described the basic elements used when creating templates. Those are constants, objects and functions.

Constants

Constants are used to create paths to files and URLs. SpinPike constants are called through the \$smarty variable.

Example: Output of the site address: {\$smarty.const.DOMAIN}

Constants available:

- . `UID` – unique user ID (is used for the system to identify the user; should be inserted in the static links, etc.)
- . `DOMAIN` – site address (`http://www.example.com`)
- . `PREFIX` - path to the SpinPike™ system root (`/home/www.example.com`)
- . `DOCUMENTROOT` - path to the site pages (`/home/www.example.com/html`)
- . `TMP` – temporary directory.
- . `TIMEZONE` - site time zone
- . `SOFTWARENAME` - name of the system (for internal use)
- . `AUTHTIMEOUT` - system timeout in seconds

Variables

Variables are used to output keywords, description, site title, etc.

\$Site

This variable contains an array of the objects and variables that define the state of the current site page (template).

Example:

```
{{$Site.User->name}} // site user name
{{$Site.modules[0]->name}} // name of the first module within the array of modules
```

Variables available:

```
modules - array “Modules” objects
Version - current version object
keywords - site keywords (META)
description - site description (META)
Module - current module object
Versions - array of the versions in the systems
administrators - array of the “System administrators” objects
```

\$Section

This object is the object of the current section. The object properties depend on the module currently used.

Properties available:

```
{ $Section->id } // section ID
{ $Section->name } // section name
{ $Section->header } // section headline
{ $Section->sort } // sorting index
{ $Section->keywords } // section keywords (META)
{ $Section->description } // section description (META)
{ $Section->rel } // parent section ID
{ $Section->Version } // section object "Version"
{ $Section->Template } // section object "Template"
{ $Section->Admin } // section object "Administrator"
```

\$Item

It is the object of the current product. It is used to manipulate product details. It is initialized when the product is selected. This object goes with the following properties:

```
{ $Item->id } // product identifier
{ $Item->name } // product name
{ $Item->description } // short description of the product
{ $Item->price } // product price
{ $Item->delivery_price } // product delivery cost
{ $Item->delivery_term } // product delivery terms
{ $Item->Image1 }, { $Item->Image2 }, { $Item->Image3 } // product pictures (objects)
{ $Item->Enode } // product section (object)
```

\$Cart

it is the object of the shopping cart. It is used to manage the items selected by the buyer.

```
{ $Cart->Goods } // product objects array
{ $Cart->sum } // total sum for the ordered items
```

Objects

You can use objects mainly to create your own templates functions. Besides, they possess such properties that can be already called in the existing functions. If you need to output a particular property of the object and you are not sure, whether it is available and how it is named, please refer to this section.

Module

Designed for system modules management.

Properties:

```
id => identifier // number
name => name // string
type => type // string
```

Application Example:

Output of the module name:

```
{ $Module->name }
```

Designed for languages management.

Language

Properties:

```
id => identifier // number
name => name // string
code => language code (by country) // string
charset => character set // string
locale => locale (Unix style) // string
Module => Module object // see "Module" object for details
```

Application Example:

Output of both of the language and module name:

```
{ $Language->name }
{ $Language->Module->name }
```

Version

Designed for versions management.

Properties:

```
id => identifier // number
name => name // string
abbr => abbreviation // string
header => header (typically used for the "title" HTML tag) // string
keywords => keywords // string
description => description // string
deflt => default version // 1 or 0
Module => Module object // see "Module" object for details
Image => Image object // see "Image" object for details
Lang => Language object // see "Language" object fore details
href => version hyperlink // string
```

Application Example:

Output of a hyperlink leading to the site version

```
<a href="{ $Version->href }">{ $Version->name }</a>
```

Template

Designed for templates management.

Properties:

```
id => identifier // number
Version => Version object // see "Version" object for details
name => name // string
src => source file // string
Module => Module object // see "Module" object for details
```

User_Group

Designed for site users group management.

Properties:

```
id => identifier // number
name => name // string
code => group code // string
```

User

Designed for site users management.

Properties:

```
id => identifier // number
name => name // string
login => login // string
mail => e-mail // string
groupid => ID of a group to which the user belongs
```

Administrator

Designed for site administrators management.

Properties:

```
id => identifier // number
name => name // string
mail => e-mail // string
root => root administrator // 1 or 0
```

Application Example:

Output of the administrator name. `{\$Admin->name}`

Content

Designed for content sections management.

Properties:

```
id => identifier // number
sort => sorting index // number
rel => parent section identifier // number
Image => Image object // see "Image" object for details
Version => Version object // see "Version" object for details
Template => Template object // see "Template" object for details
```

Admin => Admin object // see "Administrator" object for details
name => name // string
header => header (HTTP META) // string
date => date (typically Unix Timestamp) // number
date_allow => allow subsections sorting by date // 1 or 0
description => description (HTTP META) // string
keywords => keywords (HTTP META) // string
visibility => hidden section // 1 or 0
groupid => group access ID (if set to 0 - free access)
Module => Module object // see "Module" object for details
href => hyperlink leading to a section // string

Application Example:

Output of a hyperlink leading to the section; section name; section creation date:

```
<a href="{ $Section->href }">{ $Section->name }</a><br>
{ $Section->date | date_format }<br><br>
```

Content Item

Designed for pages content management.

Properties:

```
id => identifier // number  
name => name // string  
text => page content // string  
sort => sorting index // number  
Version => Version object // see "Version" object for details  
rel => parent section ID ("Content" object) // number
```

Application Example:

Output a link list at current section pages:

```
{ a_section_content_items assign="items" }  
{ foreach from=$items item="it" }  
<a href="{ $it->href }">{ $it->name }</a><br>  
{ /foreach }
```

Image

Designed for images management.

Properties:

```
id => identifier // number  
name => name // string  
width => width // number  
height => height // number  
src => name (name of the source file physically located on a hard disk) //  
string  
type => extension (extension of the source file physically located on a  
hard disk) // string  
Module => Module object // see "Module" object for details  
string => generated HTML string (<img  
rc="/pict/83c677f175e7d024fda7a3858eaf30b3.jpg" width="1200"
```

```
height="801" border="0">)
```

Application Example:

Output of an image:

```
src"},"{"$Image->type}" width=100  
height=230>
```

File

Designed for files management.

Properties:

```
.id => identifier // number  
.name => name // string  
.filename => file source name (when uploaded to the server) // string  
.src => name (name of the source file physically located on a hard disk)  
// string  
.date => file creation date // string  
.size => file size in bytes // number  
.type => MIME TYPE  
.groupid => group access ID (if set to 0 - free access)
```

Log

Designed for log files management.

Properties:

```
.id => identifier // number  
.string => notification string // string
```

“Item” object / Item

This object is used to manipulate the product details. It is initialized automatically when the product is selected.

```
id => product identifier  
name => name  
description => short description  
price => price  
delivery_price => delivery cost  
delivery_term => delivery terms  
Image1, Image2, Image3 => product images (objects)  
ENode => product section (object)
```

“Shopping cart” object / Cart

This object is used to manage the items selected by the buyer.

```
Goods => product object array  
sum => total sum for the ordered items
```

“Online payment systems” object / Paysystem

This object is used to manage the online payment systems.

```

id => payment system identifier
name => payment system name
title => payment system title (appears on the site)
class => class name to initialize the payment system class (is assigned in
the value field)

```

“Order” object / Order

This object contains the order-related information.

```

id => order identifier
pay_system => payment system
input => order placement date
status => order status (processing, confirmed, shipped)
buyer_name => buyer's name
buyer_address => buyer's address
buyer_city => buyer's city
buyer_state => buyer's state or province
buyer_country => buyer's country
buyer_phone => buyer's phone number
buyer_email => buyer's email address
buyer_zip => buyer's ZIP code
buyer_extra => additional information specified by the buyer
pay_system_data => payment system data
goods => array of objects of the purchased items

```

Template functions

The templates functions are used for creating dynamic blocks in the site templates. They are a Smarty extension (plug-ins). If you have strong knowledge of Smarty, you will be able to create your own functions. At the moment there are three classes of functions:

- . functions to work with site versions (outputting navigation by versions, version name, etc.)
- . functions to work with site sections (outputting menus, subsections lists, news, etc.)
- . functions to work with the site content (outputting section content)

Site versions functions

Version is the site versions management object. Version properties are described in the specification to the given object.

a_version

This function allows to get a site version property with the assigned ID.

Attribute	Type	Mandatory	Default	Description
get	string	no	id	the version property to be displayed
id	number	yes	n/a	version identifier

Example: Output of the version name with id=22:

```
{a_version get="name" id=22}
```

a_versions

This function allows to access site versions.

Attribute	Type	Mandatory	Default	Description
assign	string	yes	n/a	name of the versions array

Example: Output of the hyperlinks leading to all site versions available:

```
{a_versions assign="versions"}
{foreach from=$versions item="it"}
  <a href="{ $it->href}" >{ $it->name}</a><br>
{/foreach}
```

Site sections functions

Content is the site sections management object. The properties of the section are described in the specification to the given object.

a_section

This function allows to get a site section property with the assigned ID.

Attribute	Type	Mandatory	Default	Description
get	string	no	id	section property to be displayed
id	number	yes	n/a	section ID
date_format	string	yes, if the property value is "Date"	n/a	date format see the documentation provided by Smarty
module	string	no	current module	section module name

Example: Output of the section date with the format "day/month/year" and id=64:

```
{a_section id=64 get="date" date_format="%d.%m.%Y"}
```

a_sections

This function allows to access site sections.

Attribute	Type	Mandatory	Default	Description
id	number	no	current section ID	parent section ID for returned sections
assign	string	yes	n/a	name of the sections array
limit	number	no	20	maximum number of returned sections
visibility	number	no	0	hidden sections output
module	string	no	current module	name of the parent section module
static	number	no	n/a	determines whether the returned list of sections will static (fixed) *

* Suppose you need to create a menu for the News section. When you employ the `a_sections_tab` function, it may happen that the given menu is “left behind”. This is because the tab navigation will be global for each case of using the `a_sections` function. To prevent the menu from “disappearing”, the “static” parameter should be used.

Example: Output of three hyperlinks leading from the section with `id=64` to the subsections:

```
{a_sections id=64 assign="news" limit=3 module="content"}
{foreach from=$news item="it"}
<a href="{ $it->href}">{$it->name}</a><br>
{/foreach}
```

a_section_trail

This function outputs the breadcrumb trail navigation string.

Attribute	Type	Mandatory	Default	Description
current	string	no	no	outputs the name of the current section in the navigation trail
link	string	no	no	makes the navigation elements appear as links
startpage	string	no	no	name of the homepage
divider	string	no		navigation elements separator

Example: Output of the breadcrumb trail navigation, beginning with the Homepage and separated by “»”, the elements of which are hyperlinks:

```
{a_section_trail current="yes" startpage="Homepage"
divider="&raquo;"}

```

a_sitemap

This function outputs the site map as a sections tree.

Attribute	Type	Mandatory	Default	Description
limit	number	no	20	number of subsections in a branch of the site tree
totalstr	string	no	no	message to be displayed in a branch when the number of subsections exceeds the limit designated in the “limit” attribute

Example: Output of the site map with the limit in the branch set to 3:

```
{a_sitemap limit=3 totalstr="... sections in total"}
```

a_sections_tab

This function outputs the tab navigation by sections as [1-10] [11-20] [21-30].

Attribute	Type	Mandatory	Default	Description
count	number	yes	n/a	number of sections in a navigation group
rdiv	string	no	n/a	right-hand separator do not use { and } as a separator
ldiv	string	no	n/a	left-hand separator
visibility	number	no	0	output of hidden sections

Example: Output of the tab navigation with the sections grouped by 10 and separated by []:

```
{a_sections_tab count=10 rdiv="" ldiv=""}
```

a_sections_level

This function enables accessing the required subsections of the current or any other designated section. Using this function you can create menus of different nesting levels.

Attribute	Type	Mandatory	Default	Description
id	number	no	n/a	identifier of parent section for returned sections
assign	string	yes	n/a	name of the sections array
level	number	yes	n/a	subsections level
module	string	no	current module	module name for the parent section

Example: Output of the second level menu using subsections within the section with id=2:

```
{a_sections_level id=2 assign="news" level=1}
{foreach from=$news item="it"}
<a href="{ $it->href}">{$it->name}</a><br>
{/foreach}
```

a_sections_repmenu

This function outputs a replication menu that looks as follows Company |Services| Contacts

Attribute	Type	Mandatory	Default	Description
id	number	no	n/a	parent section identifier
divider	string	no		navigation elements separator

Example: Output of the replication menu using first-level sections and forward slash separator (/):

```
{a_sections_repmenu divider="/"}
```

a_sections_count

This function either outputs or assigns a variable the number of subsections within the current or any other designated section.

Attribute	Type	Mandatory	Default	Description
id	number	no	n/a	section identifier
assign	string	no	n/a	variable name to be assigned to the function output

Example:

```
{a_sections_count assign="count"}
```

a_sections_rand

This function allows accessing random subsections within the current or any other designated section. Using this function you can create an elementary internal banner rotator.

Attribute	Type	Mandatory	Default	Description
-----------	------	-----------	---------	-------------

id	number	no	n/a	parent section identifier
limit	number	yes	n/a	maximum number of returned sections
assign	string	no	n/a	variable name to be assigned to the function output

Example: Output of a random picture with a hyperlink leading to the section to which the picture is attached:

```
{a_sections_rand limit=1 assign="ad"}
{foreach from=$ad item="i"}
<a href="{i->href}">{i->Image->string}</a>
{/foreach}
```

Site content module functions

ContentItem is a site version management object. Content properties are described in the specification for the given object.

a_section_content

This function outputs the content of the current page or section.

Attribute	Type	Mandatory	Default	Description
id	number	no	n/a	section identifier
get	string	no	text	page property

Example: Output of the current page name:

```
{a_section_content get="name"}
```

a_section_content_items

This function allows to access the current section pages.

Attribute	Type	Mandatory	Default	Description
assign	string	yes	n/a	variable name to be assigned to the function output

Example: Output of the list of hyperlinks leading to the current section pages:

```
{a_section_content_items assign="items"}
{foreach from=$items item="it"}
<a href="{it->href}">{it->name}</a><br>
```

a_eshop_items

This function enables outputting of a list of products within the current section.

Attribute	Type	Mandatory	Default	Description
id	number	no	n/a	section identifier

<u>assign</u>	<u>string</u>	<u>yes</u>	<u>n/a</u>	<u>variable name to be assigned</u>
---------------	---------------	------------	------------	-------------------------------------

Example: Output of the list of links leading to the products within the section with ID=3.

```
{a_eshop_items assign="items" id=3}
{foreach from=$items item="it"}
  <a href="{ $it->href}">{ $it->name}</a><br>
{/foreach}
```

a_eshop_show_info

This function outputs a table with a list of product properties.

Attribute	Type	Mandatory	Default	Description
item	object	yes	n/a	product

Example: Output of a table with the current product properties.

```
{a_eshop_show_info item=$Item}
```

a_eshop_offer

This function enables outputting of a random array of products contained in all subsections, or within a specified section, or of the entire product catalogue.

Attribute	Type	Mandatory	Default	Description
id			n/a	section identifier
<u>assign</u>	<u>string</u>	<u>да</u>	<u>n/a</u>	<u>variable name to be assigned</u>
limit	number	нет	5	limit on the items quantity

Example: Output of the list of links leading to the products within the subsection with ID=3, and of the product short descriptions.

```
{a_eshop_offer id=3 limit=5 assign="Offer"}
{foreach from=$Offer item="it"}
  <a href="{ $it->href}">{ $it->name}</a><br>
  { $it->description}<br>
{/foreach}
```

a_eshop_add2cart

This function generates a link for adding a product to the shopping cart.

Attribute	Type	Mandatory	Default	Description
item	object	yes	n/a	product for which the link will be generated

Example: Output of the list of items within the current section with “Add to Cart” link by each item.

```
<table width="100%">
{a_eshop_items assign="items" id=$Section->id}
{foreach from=$items item="it"}
  <tr>
    <td>{ $it->Image1->string}</td>
    <td><a href="{ $it->href}">{ $it->name}</a></td>
    <td>{ $it->price}</td>
    <td><a href="{a_eshop_add2cart item=$it}">Add to Cart</a></td>
  </tr>
{/foreach}
```

```

    </tr>
  </foreach>
</table>

```

a_eshop_add2compare

This function generates a link for adding a product for comparison.

Attribute	Type	Mandatory	Default	Description
item	object	yes	n/a	product for which the link will be generated

Example: Output of the list of items within the current section with “Add for Comparison” link by each item.

```

<table width="100%">
  {a_eshop_items assign="items" id=$Section->id}
  {foreach from=$items item="it"}
    <tr>
      <td><a href="{ $it->href }">{ $it->name}</a></td>
      <td>{ $it->price}</td>
      <td><a href="{a_eshop_add2cart item=$it}">в корзину</a></td>
      <td><a href="{a_eshop_add2compare item=$it}">Add for Comparison</a></td>
    </tr>
  </foreach>
</table>

```

a_eshop_compare

This function generates a product comparison table. The table can be styled with the help of the .group-table selector, whereas the table cells can be styled using the .group selector.

Attribute	Type	Mandatory	Default	Description
deletetitle	string	no	delete	label text by the checkbox
pricetitle	string	no	Price	title «Price»
pricecurrency	string	no	\$	currency
submittitle	string	no	Delete selected items	button label

Example: Output of the product comparison table.

```

{a_eshop_compare deletetitle="delete" pricetitle="Price"
pricecurrency="Australian dollar" submittitle="Delete selected items"}

```

a_eshop_paysystems

This function generates and returns an array of the available online payment systems.

Attribute	Type	Mandatory	Default	Description
assign	string	yes	n/a	variable name to be assigned

Example: Output of the form to select a payment method.

```

<form action='/version/{ $Site.Version->abbr }/paysystem/{ $smarty.const.UID }'
method=post>

```

```

Payment method:<br>
{a_eshop_paysystems assign="Paysystems"}
{foreach from=$Paysystems item="PSys"}
  {$PSys->title}: <input type=radio name=paysystem value="{PSys-
>class}"><br>
{/foreach}
<input type=submit value="Next">
</form>

```

Integrating modules

To create a product catalogue with the possibility of buying products online, you need to do the following:



1. Create a section - "Product catalogue". It will be powered by the Catalogue module.
2. Create the properties groups and the properties within these groups for the showcased products.
3. Create subsections in the Product catalogue adding to them the required products (you will need to create at least several test sections and items during the development stage).
4. Activate a payment method (for instance, cash payment upon delivery of the product).
5. Create all the necessary templates (Product list, Product card, Shopping cart, Product comparison, Buyer details form, Thank you page, etc.).

Detailed description of how to create a Product Catalogue, properties groups, and activate a suitable payment method, is given in the SpinPike User Manual. The template creation process is described below.

"Product list" template

This template enables outputting of a list of products within a section of the catalogue. It is coded the same as other templates, but for the central part, which is a list of products. To output the list of products, the `a_eshop_items` function is used.

The below example shows a list of products in the form of a table:

	Product	Price	
	<u>HORIZONT 37CTV670i</u> Block Noise Reduction, Sound Feedback, Flat screen.	\$ 190	Add to Cart
	<u>HORIZONT 51ctv664i-11</u> 500:1 contrast ratio, Built-in 181-Channel Tuner	\$ 224	Add to Cart

The table code will look like this:

```

<table cellpadding="0" cellspacing="0" width="100%">
<tr>
  <td></td>
  <td><b>Product</b></td>
  <td><b>Price</b></td>
  <td></td>
</tr>
<tr><td bgcolor="#DDDDDD" colspan="4"></td></tr>

<!--A dynamic block that generates a product list -->

```

```

{a_eshop_items assign="items" id=$Section->id}
{foreach from=$items item="it"}
  <tr>
    <td width="80">{$it->Image1->string}</td>
    <td width="230"><a href="{ $it->href}"><b>{$it->name}</b></a><br>{$it-
>description}</td>
    <td>{$it->price}</td>
    <td><a href="{a_eshop_add2cart item=$it}">Add to Cart</a></td>
  </tr>
  <tr><td bgcolor="#DDDDDD" colspan="4"></td></tr>
{/foreach}

</table>

```

Let's scrutinize the dynamic block. We call the `a_eshop_items` function and assign a name to the array - `assign="items"`. The ID parameter determines the section from which the items should be pulled. In our case, it is the current section. The current section ID is obtained like this: `$Section->id`. So the function will look as follows:

```
{a_eshop_items assign="items" id=$Section->id}
```

After that we create a cycle within which the product list will be formed and cycle through the `$items` array. A separate array element will be named `$it` (`item="it"`).

Next, a table will be created. All the attributes that can be used when forming the product list are described in the "Item" object chapter. We output a thumbnail image of the product in the first table cell: `{ $it->Image1->string}`. The second cell contains the product name and a link leading to its detailed description

```
<a href="{ $it->href}"><b>{$it->name}</b></a>
```

Next, a short description of the product is output: `{ $it->description}`. We output the price for the product in USD in the third cell: `{ $it->price}`. The fourth cell contains an "Add to Cart" link allowing a buyer to start the buying process at once. The link is formed with the help of the `a_eshop_add2cart` function:

```
<a href="{a_eshop_add2cart item=$it}">Add to Cart</a>
```

After that the appropriate styles for the links and the text should be written. Using the Item attributes you can output the product list the way you like it, without any restrictions.

"Product card" template

This template usually contains a detailed product description, its large image, and an "Add to Cart" link. To output the product description, the `a_eshop_show_info` function is used. To output description for the current product, the below code should be added:

```
{a_eshop_show_info item=$Item}
```

The below code outputs a medium size image of the product (`Image2`), and an "Add to Cart" link, and a list of product properties:

```

<table width="100%">
  <tr valign="top">
    <td>{$Item->Image2->string}<br><br>
    <a href="{a_eshop_add2cart item=$Item}">Add to Cart</a>
  </td>
  <td>
    <div id="properties">
      {a_eshop_show_info item=$Item}
    </div>
  </td>
</tr>
</table>

```

The `a_eshop_show_info` function outputs the product properties in the form of a table:

```

<table cellpadding="5" cellspacing="2" border="0" width="100%">
  <tr>
    <td colspan=2 class=group>Dimensions</td>
  </tr>
  <tr>
    <td width=70%>Weight</td>
    <td width=30%>0,001</td>
  </tr>
  . . .

```

The properties groups have a CSS class – .group, for which you can write styles. For instance, if you want the names of the properties groups to appear semi-bold in grey, you need to write styles in the style sheet as shown below:

```
.group {color: #666; font-weight: bold}
```

To be able to modify such table, PHP knowledge is required.

“Shopping cart” template

This template enables outputting of a virtual shopping cart. It is stored as cart.tpl in the usr/author/tmpl/version_X/cart/ directory. The shopping cart template consists of two parts: an ordered items form and a payment method selection form. A typical shopping cart would look like this:

Product	Price	Delivery	Quantity	Delete
HORIZONTAL 37CTV670i	\$190	\$0	<input type="text" value="2"/>	<input type="checkbox"/>
HORIZONTAL 51ctv664i-11	\$224	\$0	<input type="text" value="1"/>	<input type="checkbox"/>
Total:		\$604		

Payment:

Cash:

Credit Card:

Let’s see how these forms are implemented: The first form in the picture above is the form outputting a list of the ordered items. It consists of four components: a table title, a dynamically generated items list, a total amount calculator, and a Refresh button. It is necessary that the <form> tag look as follows:

```

<form action='{$smarty.server.PHP_SELF}?{$smarty.server.QUERY_STRING}&
{$smarty.const.UID}' method=post>

```

After that the table is created. The table title is coded in HTML as follows:

```

<table cellpadding="6" cellspacing="0" width="100%">
  <tr>
    <th>Product</th>
    <th>Price</th>
    <th>Delivery</th>
    <th>Quantity</th>
    <th>Delete</th>
  </tr>

```

During the next step the product list should be generated. To do this, cycle through each element (product in the cart) in the array. The “Cart” object has a “Goods” attribute, which contains a product objects array. This attribute should be used in the cycle:

```
{foreach from=$Cart->Goods item="item"}
```

Now the table rows should be formed. In the first three cells the product name, its price and delivery cost should be output. In the same way the catalogue items list is generated:

```
<tr>
  <td>{$item->name}</td>
  <td>{$item->price}</td>
  <td>{$item->delivery_price}</td>
```

Next thing, a field for adding the required product quantity should be output:

```
<td><input type=text name='quantity[{$item->id}]' value='{$item->quantity}' size=3 class=input></td>
```

Then a checkbox for removing the specified items from the shopping cart should be created:

```
<td><input type=checkbox name='del[]' value='{$item->id}'></td>
</tr>
{/foreach}
```

That is the way the list of items placed in the shopping cart is generated. To output the total sum of the ordered items, the “sum” attribute of the “Cart” object is used:

```
<tr>
  <td>Total:</td>
  <td><b>{$Cart->sum}</b></td>
</tr>
```

The following code will output the “Submit” button and a mandatory hidden field:

```
</table><br>
<input type="submit" value="Refersh" class=submit>
<input type=hidden name=action value=edit>
</form>
```

After the styles have been written, the form will look as shown in the picture above.

The payment method selection form is less complicated to code, because it outputs only a list of payment methods. The payment methods are enabled through the administration interface. It is required that the <form> tag look as follows:

```
<form action='/version/{$Site.Version->abbr}/paysystem/?{$smarty.const.UID}'
method=post>
```

To output the “Payment” title, code as follows:

```
<span class="title">Payment:</span><br><br>
```

Next step, the table with the list of payment methods is generated. To output the payment methods, the `a_eshop_paysystems` function is used:

```
<table cellpadding="0" cellspacing="0">
  {a_eshop_paysystems assign="Paysystems"}
  {foreach from=$Paysystems item="PSys"}
  <tr>
    <td>{$PSys->title}: </td>
    <td><input type=radio name=paysystem value='{$PSys->class}'></td>
  </tr>
  {/foreach}
</table>
```

We call the `a_eshop_paysystems` function, which returns an array of objects. This array is processed in the foreach cycle. The first table cell will output the title of the payment method, the second – a radio button to select the required payment method.

The entire code of both forms will look as shown below:

```
<!--Form No. 1 (ordered items list)-->
```

```

<form action='{$_smarty.server.PHP_SELF}?{$_smarty.server.QUERY_STRING}&
{$_smarty.const.UID}' method=post>
<table cellpadding="6" cellspacing="0" id=cart width="100%">
<tr>
  <th>Product</th>
  <th>Price</th>
  <th>Delivery</th>
  <th>Quantity</th>
  <th>Delete</th>
</tr>
{foreach from=$Cart->Goods item="item"}
<tr>
  <td>{$item->name}</td>
  <td>{$item->price}</td>
  <td>{$item->delivery_price}</td>
  <td><input type=text name='quantity[{$item->id}]' value='{$item->
quantity}' size=3 class=input></td>
  <td><input type=checkbox name='del[]' value='{$item->id}'></td>
</tr>
{/foreach}
<tr>
  <td>Total:</td>
  <td><b>{$Cart->sum}</b></td>
</tr>
</table><br>
<input type="submit" value="Refresh" class=submit>
<input type=hidden name=action value=edit>
</form>

<!--Form No. 2 (payment method selection)-->

<form action='/version/{$_Site.Version->abbr}/paysystem/{$_smarty.const.UID}'
method=post>
<span class="title">Payment:</span><br><br>
<table cellpadding="0" cellspacing="0">
{a_eshop_paysystems assign="Paysystems"}
{foreach from=$Paysystems item="PSys"}
<tr>
  <td>{$PSys->title}: </td>
  <td><input type=radio name=paysystem value='{$PSys->class}'></td>
</tr>
{/foreach}
</table><br>
<input type=submit value="Next >" class=submit>
</form>

```

When creating a shopping cart you may just copy and paste this code to further modify it as required.

Another important note: If apart from a shopping cart you want to output on the page a menu of site sections, you should specify a module type in such functions as `a_sections`, `a_section`, etc.:

```
{a_sections id=0 assign="menu" module="Content"}
```

Without this supplement the menu will fail and display an error message. Please pay special attention to this fact.

“Product comparison” template

To be able to compare products, you need to:

1. Create an “Add for comparison” link in the product list;
2. Create a “Product comparison” template.

How to create a link: To insert a link, the `a_eshop_add2compare` function, to which a product object is passed, should be used. For instance, if you want to insert an “Add for comparison” link in a column of the table containing products, you should add the following string:

```
{a_eshop_items assign="items" id=$Section->id}
{foreach from=$items item="it"}
<tr>
  <td width="80">{$it->Image1->string}</td>

```

```
  |
```

After that, when a visitor clicks on “Add for comparison”, the indicated product will be added to a special comparison section, and the visitor will be taken to the template-driven product comparison page. The template is stored in the `usr/author/tmp/version_X/compare/` folder as `compare.tmp`.

To generate a product comparison table, the `a_eshop_compare` function is used. It is a compound function that has several parameters:

```

deletetitle – a label text by the checkbox for deleting a product from the comparison table.
pricetitle – a title for the product price column.
pricecurrency – a currency symbol to appear next to the product price.
submittitle – a label text for the button for deleting a product from the product comparison table.

```

Typically, the function will be called like this:

```

{ a_eshop_compare deletetitle="Delete" pricetitle="Price" pricecurrency="$"
submittitle="Delete selected items" }

```

The function generates a table, which can only be formatted with CSS. For this purpose specific classes are used in the table:

```

compare_submit – a class applicable to the button for deleting products;
group-table – a class applicable to the entire table;
group – a class applicable to those table cells that output the names of the product properties groups;
catimg – a class applicable to the product image;
cattext – a class applicable to the product name;
diff – a class applicable to the differing parameters (allows to single out the differing product properties).

```

The above listed classes allow flexibility in table styling. The `<table>` tag itself is generated with the following attributes:

```

<table cellpadding="5" cellspacing="2" border="0" class="group-table"
width="100%">

```

For example, if you write styles as follows

```

compare_submit {font: 10px Verdana; border: 1px solid #666; background: #F90}
group-table TD {background: #EEE}
TD.group {background: #CCC; font-weight: bold}
diff TD {background: #9F9}

```

then the product comparison table will look as shown in the picture below:

Delete selected items

	 HORIZONT 37CTV670i Delete <input type="checkbox"/>	 HORIZONT 51ctv664i-11 Delete <input type="checkbox"/>
Price	190 \$	224 \$
Dimensions		
Weight	21	54
Height	235	524
Width	214	322
Изображение		
Flat Screen	+	+
Format	16:9	9:6
Color	+	+

All in all, to design the product comparison template you only need to insert the `a_eshop_compare` function and write the appropriate styles.

“Buyer details” form; “Thank you” page

When you install SpinPike, all the necessary templates are generated automatically. However, they still need to be slightly configured in terms of design. For instance, there are four templates to process an order that will be paid by cash: `index.tpl`, `orderok.tpl`, `mailadmin.tpl`, and `mailbuyer.tpl`, which are stored in the `usr/author/tmpl/version_X/paysystem/storefront/` directory.

The `index.tpl` template contains a Buyer details form to be filled in by the buyer. The `orderok.tpl` template has a message notifying that the order has been placed successfully:

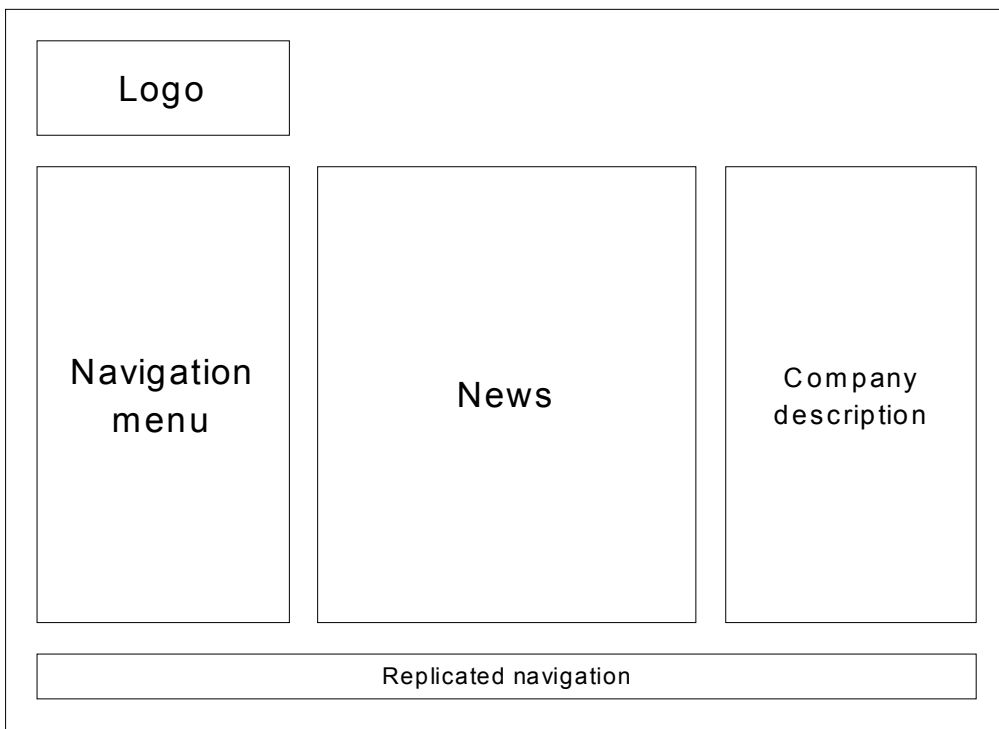
```
{foreach from=$Site.Log item="mess"}
{$mess->string}<br>
{/foreach}
```

The `mailadmin.tpl` and `mailbuyer.tpl` templates are used for creating messages that will be emailed to both the e-shop administrator and the buyer.

For orders paid via different payment methods different templates with corresponding technical documentation are used.

Template creation example

The next example will illustrate how to create a template for the homepage of a simple site. The sketch of the template is shown in picture 1.



Pic. 1. A sample layout of the site homepage

We shall not go into details of coding this template in HTML or designing and making it into page. We shall only give a simplified version of the given template coding:

```

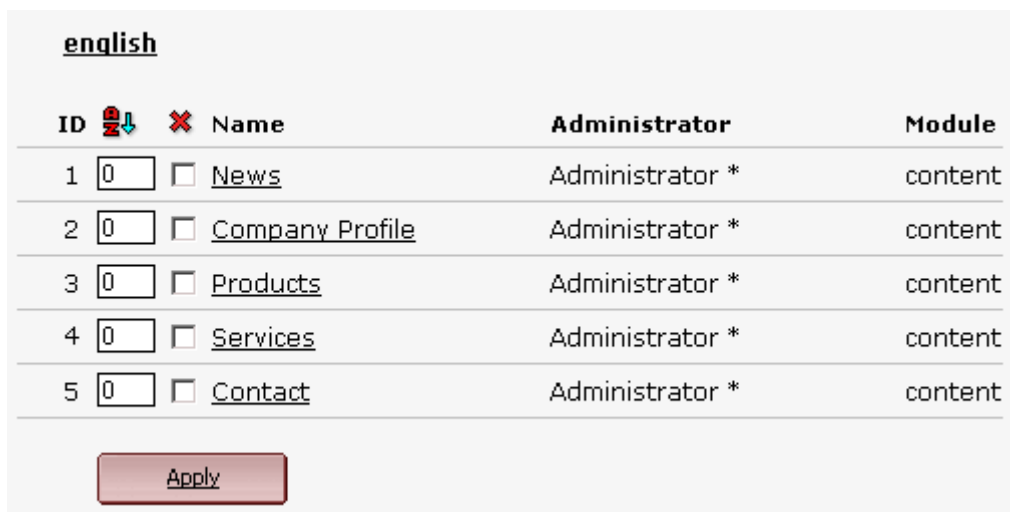
<html>
<head>
  <title>sample template</title>
  <meta name="description" content="">
  <meta name="keywords" content="">
</head>
<body>
<div id="logo">
<!-- Logo -->
</div>
<table>
  <tr>
    <td>
      <div id="mainmenu">
        <!-- main menu -->
      </div>
    </td>
    <td>
      <div id="news">
        <!-- News -->
      </div>
    </td>
    <td>
      <div id="maininfo">
        <!-- Information block on the homepage-->
      </div>
    </td>
  </tr>
</table>
<div id="altnav">
<!-- Replicated navigation -->
</div>
</body>
</html>



```

Here in <!-- Comments -->, HTML code should be inserted. Before you incorporate dynamic functions in the template, you should create a basic site structure first. For instance, the structure may look as follows:

- News
- Company Profile
- Products

To create such structure, you should go to the admin area. Detailed guidelines on how to build a site structure are given in “User Manual”. When creating a section, SpinPike assigns it an ID that cannot be changed. If you create section in the above indicated order, inside the admin area they will appear as shown in picture 2:



english				
ID		 Name	Administrator	Module
1	<input type="text" value="0"/>	<input type="checkbox"/> <u>News</u>	Administrator *	content
2	<input type="text" value="0"/>	<input type="checkbox"/> <u>Company Profile</u>	Administrator *	content
3	<input type="text" value="0"/>	<input type="checkbox"/> <u>Products</u>	Administrator *	content
4	<input type="text" value="0"/>	<input type="checkbox"/> <u>Services</u>	Administrator *	content
5	<input type="text" value="0"/>	<input type="checkbox"/> <u>Contact</u>	Administrator *	content

Pic. 2. A list of sections in the admin area

We want the news headlines, which are subsections within the News section, to appear on the homepage. Enter several news texts to see how they will be displayed. In the left-hand panel, a menu containing the entire sections of the first level will appear, that is 5 items.

Entering the site name and META-information

The name of the site is assigned when adding a site version. The `$$Site` object is used to output the site-related information. This object has a number of variables (see “Templates Functions” for details). To output the site name, the following syntax should be used: `$$Site.Version->header`. The keywords and description have a less complicated syntax: `$$Site.keywords` and `$$Site.description`.

The code will look as follows:

```
<title>{$$Site.Version->header}</title>
<meta name="description" content="{$$Site.description}">
<meta name="keywords" content="{$$Site.keywords}">
```

Creating the main menu

The menu is output with the help of the `a_sections` function. The sample menu will be created in the form of a list made of text without any images. For this purpose insert the following code in the main menu block:

```
<div id="mainmenu">
<!--Main menu -->
<ul>
{a_sections assign="menu" id=0}
{foreach from=$menu item="it"}
  <li><a href="{ $it->href}">{$it->name}</li>
{/foreach}
</ul>
</div>
```

The `id=0` parameter indicates that the root sections should be output, whereas the `assign="menu"` parameter is a service parameter, which is the name of the sections array (it should be relayed to the foreach cycle).

Please pay special attention to how the cycle is created. It is done with the help of the foreach construct to which the array name is relayed and the array element name is indicated. The elements can differ. Everything put between {foreach} and {/foreach} will be output as many times as there are elements in the array. In our case, the array contains 5 sections. Based on this the generated code will be as follows:

```
<div id="mainmenu">
<!--Main menu -->
<ul>
<li><a href="/version/ru/content/page_1.html">News</li>
<li><a href="/version/ru/content/page_2.html">Company
Profile </li>
<li><a href="/version/ru/content/page_3.html">Products</li>
<li><a href="/version/ru/content/page_4.html">Services</li>
<li><a href="/version/ru/content/page_5.html">Contact</li>
</ul>
</div>
```

Outputting news headlines

To output subsections of the given section, it is necessary to know its ID. Each piece of news is a subsection within the News section with the id=1. To output the news headlines, we shall use the a_sections function. Suppose we want to output 5 news headlines. Then the code will look as follows:

```
{a_sections id=1 assign="news" limit=5}
{foreach from=$news item="i"}
  <a href="{ $i->href }">{ $i->name}</a>
  /{ $i->date|date_format:"%d.%m.%Y"}/<br>
  { $i->header}
  <br><br>
{/foreach}
```

That is the output elements will be a news title, which is simultaneously a hyperlink, a news date, and a headline. The code will be as follows:

```
<a href="/version/ru/content/page_11.html">Second news text
title</a> /11.02.2003/<br>Second news text headline<br><br>
<a href="/version/ru/content/page_10.html">First news text
title </a> /21.03.2003/<br>First news text headline
<br><br>
```

Next comes the Company Profile. The company-related information (a right-hand block) makes the homepage content. To output the content, the a_section_content function is used. In our case, the code that will output this block will be as follows:

```
<div id="maininfo">
<!-- Information block on the homepage-->
{a_section_content}
</div>
```

Inserting a replicated navigation

The replicated navigation is output by the a_sections_repmenu function. The code will look as follows:

```
<div id="altnav">
<!--Replicated navigation -->
{a_sections_repmenu divider="|"}
</div>
```

As you could see, embedding dynamic functions in a template does not take much time. Although, the template we used as an example was not difficult to implement. Such templates are commonly used in web site development.

Judging from experience, we can say that after a short training creating dynamic templates on the basis of static HTML pages will not take longer than 30 minutes per template.

The final template code will be as follows:

```
<html>
<head>
  <title>{$Site.Version->header}</title>
  <meta name="description" content="{$Site.description}">
  <meta name="keywords" content="{$Site.keywords}">
</head>
<body>
<div id="logo">
<!-- Logo -->
</div>
<table>
  <tr>
    <td>
      <div id="mainmenu">
<!-- Main menu -->
      <ul>
        {a_sections assign="menu" id=0}
        {foreach from=$menu item="it"}
          <li><a href="{it->href}">{$it->name}</li>
        {/foreach}
      </ul>
    </div>
    </td>
    <td>
      <div id="news">
<!-- News -->
        {a_sections id=1 assign="news" limit=5}
        {foreach from=$news item="i"}
          <a href="{i->href}">{$i->name}</a> /
          {i->date|date_format:"%d.%m.%Y"}/<br>
          {i->header}
          <br><br>
        {/foreach}
      </div>
    </td>
    <td>
      <div id="maininfo">
<!-- Information block on the homepage -->
        {a_section_content}
      </div>
    </td>
  </tr>
</table>
<div id="altnav">
<!-- Replicated navigation -->
  {a_sections_repmenu divider="|"}
</div>
</body>
</html>
```

Picture 3 shows the way the page with the sample content will appear in a browser window.

The dynamic templates that have been created should be added to the system through the administration interface. Each site version should have an individual set of templates.

The templates are stored on the server in special catalogues.

Once the templates are added to the system, they are placed in the `_engine/usr/spinpike/tmpl/version_#/content` content directory. You can easily edit the templates without using the web interface.



Pic. 3. The sample homepage in a browser window

* If you have a question that has not been touched upon in the given document, please contact our technical support service at support@savvybox.com.