VirtueMart Developer Manual

Soeren Eberhardt-Biermann

VirtueMart Developer Manual

Soeren Eberhardt-Biermann

Table of Contents

List of Figures

Preamble

The content of this document is related to VirtueMart.

VirtueMart is free Software, licensed under GNU/GPL; VirtueMart [http://www.virtuemart.net]

The Software 'VirtueMart' is intended for use in Joomla! and Mambo (versions 4.5.1 - 4.6.2). Joomla! or Mambo are required for running VirtueMart.

(Joomla / Mambo is free Software, licensed under GNU/GPL)

*The abbrevation **VM**, which stands for VirtueMart, is used in this document.*

Chapter 1. Introduction

1.1. History

VM has its roots in an Open Source Shopping Cart Script called *phpShop*. This script was developed by Edikon Corp. and the phpShop community (see www.phpshop.org [http://www.phpshop.org]).

VM was forked from phpShop version 0.8.0 at the end of the year 2003. It was maintainend and developed under the name mambo-phpShop until September 2005. In October 2005 it was renamed to VirtueMart.

1.2. Differences to phpShop

VM still contains some code parts from phpShop, but experienced phpShop coders will find similarities.

So when you have experience with phpShop or you are to integrate an existing Add-On for phpShop into VM, you will have to know what is the difference between both scripts.

1. **Parameter Renames/Changes**

   VM has introduced several new parameters and parameter name changes.

   page — Syntax Change Most important is the change of the page parameter syntax from a pattern like "shop/index" to "shop.index" just to provide support for Search Engine Friendly (SEF) links in your Joomla site. All references to the paramter page that contain a slash instead of a dot will not be recognized and VM will print out "module not found" error.

   offset — Outdated/removed The offset parameter was completely replaced by the parameter "limitstart", which is Joomla standard for page navigation. Although there's a global workaround to fill $offset with the value of $limitstart it's not recommended to work with offset.

   limitstart — The limitstart parameter is the replacement for offset and can be used just like this.

   Itemid — This parameter is new and not VM-specific. It's a mandatory parameter that tells Joomla, which Menu Item is selected and active, so the pathway can be written correctly (Home -> Online-Shop) and modules which shall only be shown on specific pages are hidden/shown.

2. **Database Interface**

   phpShop has its own database class: ps_DB, in a file called db_mysql.inc. This database class has been completely modified to be a wrapper class for the Joomla Standard Database Class 'database'. The new filename is `ps_database.php`. The class name is still ps_DB, but its a Child Class of the Joomla database class (class ps_DB extends database) and inherits all methods and properties. This has a lot of advantages: the class is safe against Joomla database class changes and it provides backward compatibility for the masses of database calls and queries in the scripts (which don't use the Joomla functions, but the phpShop functions!). VM doesn't connect to the database, but it uses the connection Joomla has built up. This is for optimal performance since VM doesn't connect to the database each time a query is to be run.

3. **Database Structure**

Table names have changed and got a prefix!! Use `#__{vm}_tablename` instead of tablename. The #__ stands for the dynamic Joomla table name prefix. The `{vm}` stands for the dynamic table name prefix of VM which allows to have more than one shop in one database.

The database structure of phpShop had to be changed, because Joomla provides an excellent framework with session handling and user management. The following tables have been removed:

- auth_user_md5 (jos_users stores passwords)
- intershipper
- language
- sessions

There have been added several tables: jos_pshop_affiliate, jos_vm_affiliate_sale, jos_vm_creditcard, jos_vm_manufacturer, jos_vm_manufacturer_category, jos_vm_product_download, jos_vm_product_mf_xref, jos_vm_product_reviews, jos_vm_product_votes, jos_vm_shipping_carrier, jos_vm_shipping_rate, jos_vm_visit, jos_vm_waiting_list, jos_vm_zone_shipping.

4. **Session handling**

   Joomla provides a framework with session handling - no need to have an own session class! No hidden_session() calls are needed anymore. The existing session class has become the global link formatter! The functions url and purl are needed to format links SEF or append the Itemid parameter.

5. **Separation into component and modules**

   A Joomla site consists of various elements like components, modules, templates and Mambots - most likely you will know components, modules and templates. A Component is the Main Part of the Page in the "Main Body". Can be installed/uninstalled trough the Component Manager and have their own configuration/interface. Modules are sideblocks surrounding the Main body. They can be installed/uninstalled and configured using the Module Manager. The Main application "VirtueMart" is run in the component part. The Component contains all core files. The module "mod_virtuemart" was written to provide all important links so the component can be controlled: Category links, Mini-Cart, Product Search Form, Account Maintenance, Admin.

1.3. Joomla! Integration

The Joomla Integration of VM is very special, because of its origin. It doesn't completely comply to Joomla's Component Coding Standards. VM uses some own functions for database access, page navigation, search and listings. By using old code from phpShop, this little bit of compatiblity can be maintained (so one can integrate extensions written for phpShop).

Chapter 2. Basics

2.1. Directory and File Structure

VM holds most of its files in the `/administrator` part of Joomla. The only files stored in the `/components` part of a Joomla site are those, which must be accessible from the Frontend of a Joomla site, even when the Administrator part is secured by htaccess files.

/administrator/components/com_virtuemart/	Contains file for the administration interface of VM. Because the administrative interface is also accessible from the frontend, those files are not restricted to the Joomla! Coding Standards. Important files: • `header.php` (Code for the Drop-Down Menu of the administration) • `virtuemart.cfg.php` (central Configuration File) • `toolbar.virtuemart.html.php` (Standard Toolbar - Joomla! style) • `toolbar.html.php` (Toolbar for ExtJS in extended Layout)
/administrator/components/com_virtuemart/classes/	Holds all the core classes which are used by VM Important: • `ps_database.php` (wrapper for Joomla's database object $database) • `ps_cart.php` (controls the cart contents) • `ps_main.php` (not a class, contains central functions, e.g. for image upload) • `ps_session.php` (basic session management, URL formatting)
/administrator/components/com_virtuemart/classes/Log/	Contains a slightly modified version of PEAR's Log class
/administrator/components/com_virtuemart/classes/shipping/	Contains Shipping Modules & their informational Files
/administrator/components/com_virtuemart/classes/payment/	Contains Payment Modules & their informational Files
/administrator/components/com_virtuemart/classes/pdf/	Contains the classes of the HTML2FPDF Package (see sourceforge.net/projects/html2fpdf [http://sourceforge.net/projects/html2fpdf])
/administrator/components/com_virtuemart/classes/phpInputFilter/	contains the phpinputfilter class for VirtueMart
/administrator/components/com_virtuemart/classes/phpmailer/	

Contains the classes of the phpMailer Package (also used by Joomla! and Mambo) - see phpmailer.sourceforge.net/ [http://phpmailer.sourceforge.net/].

/administrator/components/com_virtuemart/html/

Holds files which are used for loading VirtueMart pages.

They are ordered by the core module name (e.g. **checkout.*.php** for the core module *checkout*)

Important files:

- `basket.php` (controls the Cart)
- `ro_basket.php` (controls the Cart on the last step of checkout, ro = read only)

/administrator/components/com_virtuemart/languages/

Contains the Language Files which are included from virtuemart_parser.php.

/administrator/components/com_virtuemart/sql/

Holds SQL Dump Files for building up the structure for the tables used by VirtueMart.

/components/com_virtuemart/

Holds the files wich are used to control the call of the Shop from the Frontend.

Important files:

- `virtuemart.php` (the file included by Joomla on a call to `index.php?option=com_virtuemart&....`)
- `virtuemart_parser.php` **(the central file for VM, prepares the session, authentication, cart & runs functions)**
- `show_image_in_imgtag.php` (used to display dynamically resized images - using the *class.img2thumb.php*)

/components/com_virtuemart/css/

Contains the shop's css file (`shop.css`) and css styles needed for the frontend administration (`admin.css`)

/components/com_virtuemart/js/

Contains Javascripts

/components/com_virtuemart/shop_image/

Images for the Shop

/availability

Contains images for displaying the availability of a product.

Tip

All images in this folder are automatically parsed and displayed in the product form

		for selection as the availability image for a product - so just copy them here.
	/category	Contains images for categories
	/product	Contains Product Images + resized product images
	/ps_image	Images for the administrative interface
	/vendor	Vendor Logos
/components/com_virtuemart/themes	Contains the themes for VirtueMart. Each theme has the following structure:	

Directory/File	Function
/admin.css	the stylesheet for the frontend administration
/theme.config.php	the configuration file for the theme
/theme.css	the main stylesheet for the theme
/theme.js	the main javascript controller for the theme
/theme.php	the "controller" for the theme; used to declare functions and include stylesheets (and more!)
/theme.xml	contains the theme configuration parameters and additional information
/images/	holds the theme images
/templates/	holds the template files that allow you to style and restructure the shop pages
/templates/basket/	holds the basket templates
/templates/browse/	holds the templates for the product listing page
/templates/common/	holds some commonly used templates, like the price display, pathway and the product snapshot
/templates/pages/	holds the templates for all other pages (e.g. 'account.order_details.tpl.php')
/templates/order_emails/	holds the order email templates as used when an order is placed
/templates/product_details/	holds the product detail templates

2.2. Main Flow Chart

2.2.1. Joomla Part

Joomla uses the variable **option** to load a specific component. This variable must have the value "com_virtuemart" to load VM. Called on the Frontend, Joomla searches the directory `/components` for a directory called *com_virtuemart* and a file called `virtuemart.php` in it.

When called in the backend, Joomla searches the directory `/administrator/components` for a directory called *com_virtuemart* and a file called `admin.virtuemart.php` in it.

If found, the file is included.

2.2.2. Shop Part

When the Shop is loaded, one of the first things is to load the file `virtuemart_parser.php` using the require_once command. It makes core interactions like the Joomla.php file /mainframe class and after that looks for a variable called **page** (can be passed by GET or POST).

The page variable consists of the pagename and the core module name:

`shop.browse` # **shop** is the name of the shop core module and **browse** is the name of the page.

Tip

Core modules are listed in the table mos_vm_modules.

Calling `index.php?com_virtuemart&page=shop.browse` in your Joomla site would let VM include the file

`/administrator/components/com_virtuemart/html/shop.browse.php`.

2.3. Core Modules & their Functions, Environment Variables

2.3.1. Core Modules

In order to ease with which new features can be added to mp, the concept of using modules has been introduced. A module defines a feature set of VM by providing class files and html layouts related to that particular module. It is very important to understand how modules work since everything, including the shop, is a module.

Each module is defined and set in the VM module register. The module definition form allows the site administrator to define the information for each module, e.g. the module name, the perms of this module and its description.

You can reach the module list in the administrative interface using "Admin" # "List Modules".

Example: The core module "product" is one entry in the table mos_vm_module. Its pages must be called using "*..&page=product.* ....". If the user has appropriate permissions, the page is loaded - if not, an error message is generated.

2.3.2. func

Each core module has a list of functions that can be executed. For example, to add a product into the system, a function called **productAdd** exists in the table `jos_vm_function`.

When you add a product, you pass the hidden variable `func` with a value of *productAdd* to the system (besides all the other form fields).

If the current user has the permissions to execute the function (permissions can be set for each function separately), the file `virtuemart_parser.php` looks for the class file name and the function name mapped in the table `mos_vm_function` for that specific function name (**productAdd**). In this case we get **ps_product** as the class name and **add** as the function name.

After having fetched this information, we can start to execute the *real* function, which is done in this part of **virtuemart_parser.php**:

```
// Load class definition file
require_once( CLASSPATH."$class.php" );
$classname = str_replace( '.class', '', $funcParams["class"]);
if( !class_exists(strtolower($classname))) {
 $classname = 'vm'.$classname;
}
if( class_exists( $classname )) {
 // create an object
 $$classname = new $classname();

 // RUN THE FUNCTION
 // $ok  = $class->function( $vars );
 $ok = $$classname->$funcParams["method"]($vars);
}
```

First, the file `ps_product.php` is loaded, then an object of the class `ps_product` is created and the function add is called on that object. The function returns `true` on success and `false` on failure. The variable `$ok` stores the function result. All this code is exectuted using the PHP eval command for creating and executing PHP code on-the-fly.

If you wonder what the variable `$vars` is: it's just a working copy of the superglobal `$_REQUEST` Array and used as the array `$d` inside of the functions.

2.3.3. Other important Environment variables

Array $cart

The current cart contents. The array has the following structure:

```
[cart] # Array (
            [idx] # 1
            [0] # Array (
                     [quantity] # 1
                     [product_id] # 10
                     [description] # Size:big; Power:100W
                   )
          )
```

In this example, the car contains one product with the quantity of 1, the product ID 10 and a description.

The index "idx" is an integer and contains the size of the cart (number of different products in it, regardless of their quantity). This variable is always available in the global `$_SESSION` array: `$_SESSION['cart']`.

Array $auth

All the user information in one Array, always available in the global `$_SESSION` array.

```
[auth] # Array (
            [show_prices] # 1
            [user_id] # 0
            [username] # demo
            [perms] #
            [first_name] # guest
            [last_name] #
            [shopper_group_id] # 5
            [shopper_group_discount] # 0.00
            [show_price_including_tax] # 1
            [default_shopper_group] # 1
            [is_registered_customer] #
          )
```

	These are the example settings for an unregistered, not-logged-in user.
ps_session $sess	Mainly used to format and print URLs for the Shop.

2.3.4. Logging events with the vmLogger object

VirtueMart allows logging events that occur during the execution of the script. The global variable `$vmLogger`, which is used for logging purposes is an object of the class `Log_display`. This class is a child class of the Log class, which is a PEAR extension.

Note

You must declare

```
global $vmLogger;
```

to be able to use this variable inside of a function.

"Logging" means to log a message to display them to the user. While a function is executed (because its execution was triggered by the variable $func) in the file `virtuemart_parser.php`, the events are buffered. When the function call has ended, the contents of the log are flushed and all messages are displayed to the user in the order they were added to the log: first in, first out.

After that implicit flushing is enabled - what means that you can log a message and it is printed into the HTML code where you call the log function.

Currently the Log_display class used by VM offers 9 log levels:

- System is unusable (**PEAR_LOG_EMERG**)
- Immediate action required (**PEAR_LOG_ALERT**)
- Critical conditions (**PEAR_LOG_CRIT**), formatted by CSS style `log_crit`
- Error conditions (**PEAR_LOG_ERR**), formatted by CSS style `log_error`
- Warning conditions (**PEAR_LOG_WARNING**), formatted by CSS style `log_warning`
- Normal but significant (**PEAR_LOG_NOTICE**)
- Informational (**PEAR_LOG_INFO**), formatted by CSS style `log_info`
- Debug-level messages (**PEAR_LOG_DEBUG**) formatted by CSS style `log_debug`
- Advice messages (**PEAR_LOG_TIP**, added for VM), formatted by CSS style `log_tip`

Please note that Debug log entries are only shown to the user, when DEBUG is enabled by configuration.

To log an event, you can use a special function for each log level:

- `$vmLogger->emerg( 'My emergency message to the user' );`
- `$vmLogger->alert( 'My alarm message to the user' );`
- `$vmLogger->crit( 'My critical message to the user' );`
- `$vmLogger->err( 'My error message to the user' ); // Mainly used to log errors in`

- `$vmLogger->warning( 'My warning message to the user' ); // Mainly used to t`
- `$vmLogger->notice( 'My Notice to the user' );`
- `$vmLogger->info( 'My informational message to the user' ); // Used to give`
- `$vmLogger->debug( 'My debug message to the user' ); // Only displayed when`
- `$vmLogger->tip( 'My advice to the user' ); // Used to display Advice messag`

2.4. Database Structure

As said before, all Tables used for VM begin with the prefix _vm_. VM doesn't use Joomla core tables for storing data.

2.5. Database Access

VM uses its own database access class for dealing with the database.

The database class file is

`/administrator/components/com_virtuemart/classes/ps_database.php`.

This database class extends Joomla's database class (class ps_DB extends database) and provides additional functions, to be able to use older phpShop code. So this class is just a wrapper class for Joomla's database object and doesn't open new connections to the database!

- Start a query: call the method `query( string $query )`

  ```
  $db->query( 'SELECT email FROM #__users');
  ```

- Get the resulting record set: call method `next_record( void )`:

  ```
  $db->next_record();
  ```

  (returns false when no result can be returned or the end of the record set has been reached)

- Fetch the value of an attribute of the record set: method `f( string $nameOfTheAttribute )`

  ```
  $db->f( 'email' );
  ```

  Alternative: method `sf( string $nameOfTheAttribute )` returns the value of the attribute specified by $nameOfTheAttribute or - when it's not available - the value of `$vars[$nameOfTheAttribute]`.

- Print (echo) the value of an attribute of the record set: method `p( string $nameOfTheAttribute )`

  ```
  $db->p( 'email' );
  ```

  Alternative: method `sp( string $nameOfTheAttribute )` prints the value of the attribute specified by $nameOfTheAttribute or - when it's not available - the value of `$vars[$nameOfTheAttribute]`.

- Get the number of returned records: method `num_rows( void )`.

  ```
  if( $db->num_rows() > 0 ) { // we have a record set! }
  ```

2.6. User Integration

VM uses Joomla's user table *jos_users* for the User Management. Users which are no customers, have just empty values in their additional customer fields in that table.

There can be users who are no customers, but there can't be customers who are no registered users on the Joomla Site.

The Shop has an own registration procedure which adds all entries for the additional user fields durch (assigning the customer to a shopper group, to a vendor...)

- `jos_users` contains BillTo Address Information
- `jos_vm_user_info` contains ShipTo Address Information (when the customer has added ShipTo Addresses)
- `jos_vm_order_user_info` contains a copy of the BillTo (&ShipTo) Address at the moment when an order is placed

Chapter 3. Modifying the Layout

The most important part of the Layout of your Shop is the Joomla template (Joomlahut.com [http://mambohut.com/] is a good start)!

3.1. Themes and Templates

Starting with version 1.1, VirtueMart offers to style the shop using themes and templates.

3.1.1. Themes

3.1.1.1. Introduction

A theme is what defines certain parts of the look and feel of your shop. All installations of VirtueMart should start out with the default theme. You can change the Theme in the "Site" section of the configuration form. Usually there's only one theme available, so you can't switch. A good way to think of themes is as "plugins", which contain a collection of page templates, images, stylesheets, javascript and other files. While editing these files can be a bit technical, the layout is separated from the content allowing you to make and distribute your own themes.

3.1.1.2. Theme Configuration

Themes can be configured for better usability. Configuration parameters can be used to turn on and off certain parts in templates. The configuration parameters are defined in the file `/components/com_virtuemart/themes/THEMENAME/theme.xml` and follow the common parameter syntax for components, modules and mambots as used in Mambo >= 4.5.1 and Joomla! (Read more about the mosinstall parameter XML syntax). When a user chooses to configure a theme (Shop Configuration # Tab "Site # "Layout", these parameters are parsed and displayed in a nice form. Now the user can make changes to the configuration. The current configuration values are stored in the file `/components/com_virtuemart/themes/THEMENAME/theme.config.php`. This file is renewed with the new configuration values each time a user saves the theme configuration.

How to use configuration values in templates

It's easy to get or set the value of a configuration parameter from inside a template. You just need to call **$this->get_cfg('parameterName')** to get the value for this parameter. Example from `/templates/product_details/flypage.tpl.php`:

```
// Show the vendor link?
if( $this->get_cfg('showVendorLink', 1)) {
    echo $vendor_link;
}
```

The second parameter let's you predefine a "default" value, which is used when the configuration file for this theme doesn't have a value for this parameter.

3.1.2. Templates

3.1.2.1. Introduction

Important

A template in VirtueMart has nothing in common with what is a "template" in Joomla!.

A template is a file holding HTML and PHP code for displaying a page or a part of it. Example: the file `/templates/product_details/flypage.tpl.php`. It is used to display details of a product.

3.1.2.2. Placeholders or Variables?

You might want to know which placeholders or variables are available to style and display certain details...well that differs from template to template. The reason for this is that all variables must be imported into the template before you can use them. Most important: there are no more placeholders like {product_name} as used in VirtueMart 1.0.x. All details are avalaible in PHP variables. So all you need to know is: which variables can be used in what template and how can I use them?

Use PHP in the template

If you want to output the value of a variable all you need to do is add PHP brackets and echo the variable inside:

```
<!-- Print out the value of a variable -->
<p>Product Name: <?php echo $product_name ?></p>
```

Don't forget to close the PHP brackets and always use valid code. If you don't it might break your site.

Predefined Global Variables

$VM_LANG This is the global language object. It displays strings which are defined in the language file for VirtueMart.

Usage:

```
<?php echo $VM_LANG->_('PHPSHOP_FLYPAGE_LBL') ?>
```

This would print "Product Details".

$sess Used to build (SEF-enabled) URLs that can be used in links.

Usage:

```
<a href="<?php echo $sess->url( $_SERVER['PHP_SELF'] .'?page=shop.product
```

Predefined Variables for the Flypage / Product Details Template

product_name	The Product Name
product_sku	The Product SKU
product_s_desc	The Product Short Description
product_description	The Product Description
product_weight_uom	The Product Weight's unit of measure
product_length	The Product Length
product_height	The Product Height
product_width	The Product Width
product_lwh_uom	The Unit of Measure for the Product Length,Width,Height

product_url	The Product URL
product_in_stock	The Number of Products currently in Stock
product_available_date	The UNIX Timestamp for the Product Availability Date
product_availability	The Product Availability String
product_special	Y or N, is the product on special?
product_discount_id	The Product's Discount ID
cdate	The Product's Creation Date (UNIX Timestamp)
mdate	The Product's last Modification Date (UNIX Timestamp)
product_sales	The Number of Sales of this Product
product_unit	The Product's Packaging Unit
product_packaging	The Number of Products per Package
product_price_lbl	The price label; "Price:"
product_price	The completely formatted product price
product_packaging	Product Packaging information
file_list	The list of additional files for this product (when the product has files assigned to it)
product_availability	The product availability information; includes the "number of products in stock" and the average delivery time
addtocart	The Add-To-Cart Button Code
product_type	Product Parameter Values
product_reviews	The List of Product Reviews
product_reviewform	The Form to post a new Product Review
product_image	The small product image; complete image tag; wrapped into an URL when available.
product_full_image	The relative filename of the product full image (relative to / components/com_virtuemart/shop_image/products/
product_thumb_image	The relative filename of the product thumbnail image (relative to / components/com_virtuemart/shop_image/products/
buttons_header	The PDF, Email and Print buttons
navigation_pathway	The pathway to the product (Power Tools # Outdoor Tools # Chain Saw)
more_images	The link to the "Product Images" page when the product has additional product images
manufacturer_link	The link to the manufacturer info page
vendor_link	The link to the vendor info page

edit_link	The link to the product form of this product (admin only)
ask_seller	The link to the "Ask a question about this product" page
related_products	The list of related products
navigation_childlist	The child categories for the current category (this product is located in)
images	The object list of all additional images this product has
files	The object list of all additional files this product has

Predefined Variables for the Browse Page / Product Listing Templates

product_name	The Product Name
product_sku	The Product SKU
product_s_desc	The Product Short Description
product_weight_uom	The Product Weight's unit of measure
product_length	The Product Length
product_height	The Product Height
product_width	The Product Width
product_lwh_uom	The Unit of Measure for the Product Length,Width,Height
product_flypage	The internal Product URL (to the details page)
product_url	The external Product URL
product_in_stock	The Number of Products in Stock
product_available_date	The UNIX Timestamp for the Product Availability Date
product_availability	The product availability information; includes the "number of products in stock" and the average delivery time
cdate	The Product's Creation Date (UNIX Timestamp)
mdate	The Product's last Modification Date (UNIX Timestamp)
product_price	The completely formatted product price
form_addtocart	The Add-To-Cart Button Code
product_rating	The average Product Rating
product_details	The string "Details"
product_full_image	The relative filename of the product full image (relative to / components/com_virtuemart/shop_image/products/
product_thumb_image	The relative filename of the product thumbnail image (relative to / components/com_virtuemart/shop_image/products/
images	The object list of all additional images this product has

files	The object list of all addtional files this product has
buttons_header	The PDF, Email and Print buttons
browsepage_header	The heading, the category description
parameter_form	The Parameter search form
orderby_form	The sort-by, order-by form PLUS top page navigation
navigation_pathway	The pathway to the product (Power Tools # Outdoor Tools # Chain Saw)
navigation_childlist	The child categories for the current category (this product is located in)
browsepage_footer	The footer with page navigation and result counter

3.1.2.3. Using and "Fetching" Templates

Create a template object

First of all, the template object must be created as an instance of the class vmTemplate.

```
//Create an object of the class vmTemplate
$tpl = vmTemplate::getInstance();
```

Import variables into the template

If you want to use a certain variable in your template you must import it before! Templates don't have the global scope. So all variables are not available in a template unless you "import" them. This can be done using the set method from the controller file (e.g. "shop.browse.php"):

Importing a variable into the template

```
## Syntax: $tpl->set( 'This_is_the_variableName_available_in_the_template',

$tpl->set( 'product_name', $product_name );
```

The variable $product_name is just an example. You must care to get the variables from the database! The browse page offers an database object holding all important product information: $db_browse. So if you need to access the "product_weight" or whatever value, you need to call $tpl->set('product_weight', $db_browse->f('product_weight')); in order to get the correct value.

Parse the template

If you have finished importing all needed variables into the template object, you can "fetch" it using this syntax:

```
// Parse a template
$contents = $tpl->fetch( 'product_details/myFlypage.tpl.php' );
// Print out the contents
echo $contents;

// Alternative: Fetch a cached template (caches if no cached copy available
$contents = $tpl->fetch_cache( 'product_details/myFlypage.tpl.php' );
// Print out those contents
echo $contents;
```

3.2. Finding the right File

When you want to modify a part of your Shop (that can't be changed in its layout using the Joomla template's CSS), you must of course know, which file you have to modify, to create the layout you want.

To quickly find the file, which produces the HTML output you're seeing, you can enable the **DEBUG mode** ("Admin" # "Configuration" # "Path & URL" # check "DEBUG?" and save.

After having done that, you will see blue info icons all over the Shop, which show the file name of the included file on mouseover.

The most changed files are

- `.../html/shop.browse` (the product listing / category overview)
- `.../html/shop.product_details.php` (the product detail page / view)
- `.../html/shop.index.php` (the default Shop Homepage (when the parameter `page` is omitted))

3.3. Modifying Templates

3.3.1. Flypage Templates

Flypage (or product details) templates can be found in `/themes/VM_THEME/templates/product_details/`.

They are loaded and filled with content in the file `/html/shop.product_details.php`.

3.3.2. Browse Templates

Browse templates define the display of a single product in the product listing. So you can only modify the contents of the boxes, which are filled with product information in the product listing of a category. The number of those "boxes" - which are displayed in a single row of the product listing - can be changed in the Category Form of that category (see *Number of Products per row*) or globally in the Shop Configuration (for the case that no category_id is passed to the Shop).

Browse (or product listing) templates can be found in `/themes/VM_THEME/templates/browse/`.

They are loaded and filled with content in the file `/html/shop.browse.php`.

3.3.3. Order Confirmation Email Templates

Order Confirmaton Email Templates define the layout of the confirmation email that is sent out to a user after having placed an order.

These Email templates can be found in `/html/themes/VM_THEME/templates/order_emails/`.

They are loaded and filled with content in the file `/classes/ps_checkout.php`, `function email_receipt()`.

The concept is to define placeholders in the template and replaced them by the real contents on load. This is done using the PHP function `str_replace`.

3.3.4. Basket Templates

Basket templates control the layout of the basket.

The templates can be found in the directory `/themes/VM_THEME/templates/basket/`.

The special about the basket is that there are four different templates: Two for displaying the Cart content including Tax (`basket_b2c.html.php` and `ro_basket_b2c.html.php`) and two for displaying the Cart content without tax (adding it afterwards) - `basket_b2b.html.php` and `ro_basket_b2b.html.php`.

b2c = Business to Customer (prices include tax)

b2b = Business to Business (prices don't include tax)

The **basket_** files are included in `/html/shop.cart.php`, `/html/basket.php` & `/html/ro_basket.php` and in the `/html/checkout.index.php` except that the **ro_basket_** file is displayed on the last step of the checkout (when the cart contents can't be modified any more - ro_basket = read only basket).

The variables which have been prefilled in `/html/basket.php` and `/html/ro_basket.php` are just printed out in the templates.

Chapter 4. Creating or modifying Extensions

Besides core modules, you can also add shipping and payment modules into VM. The concept of both - shipping and payment modules is to provide an API with a defined specification (similar to an interface), where the modules can plug themselves in. The modules implement the required functions and thus can communicate with the Shop and give their services.

4.1. Payment Modules

VirtueMart has a lot of predefined payment methods. Some of these payment methods are controlled by payment modules.

Example:

- Payment Method: “Credit Card”, Payment Module: ps_authorize

  Handles credit card authentication with the authrize.net server.

- Payment Method “Cash on delivery”, Payment Module: none (ps_payment is an empty payment module)

  No business logic needed for this kind of payment.

4.1.1. Basics

Virtuemart knows the following general payment method types:

Payment Processor	Asks the Customer for Credit Card Details and processes the payment before the order is placed. Automatic Credit Card Processors do a server-to-server communication before the order is placed (e.g. authorize.net AIM)
Credit Card	Just asks the customer for the credit card details. The details are securely stored in the database for capturing the payment manually afterwards.
Bank Debit	Asks the customer for his/her bank account details, so you can get the money from the customer’s bank account afterwards.
Address only/Cash on delivery	No payment processing by the module.
HTML form-based	Transfers the customer to another server where he/she can pay. This is done after the order has been placed (examples: PayPal, Worldpay, 2Checkout)

4.1.2. Quick-Create a new payment method

If you have form code for a form-based payment method (most payment providers use this way), you just need to select "Store" # "Add Payment Method" from the VirtueMart admin drop-down menu.

An empty payment method form opens. Now fill in the details of your payment method like this:

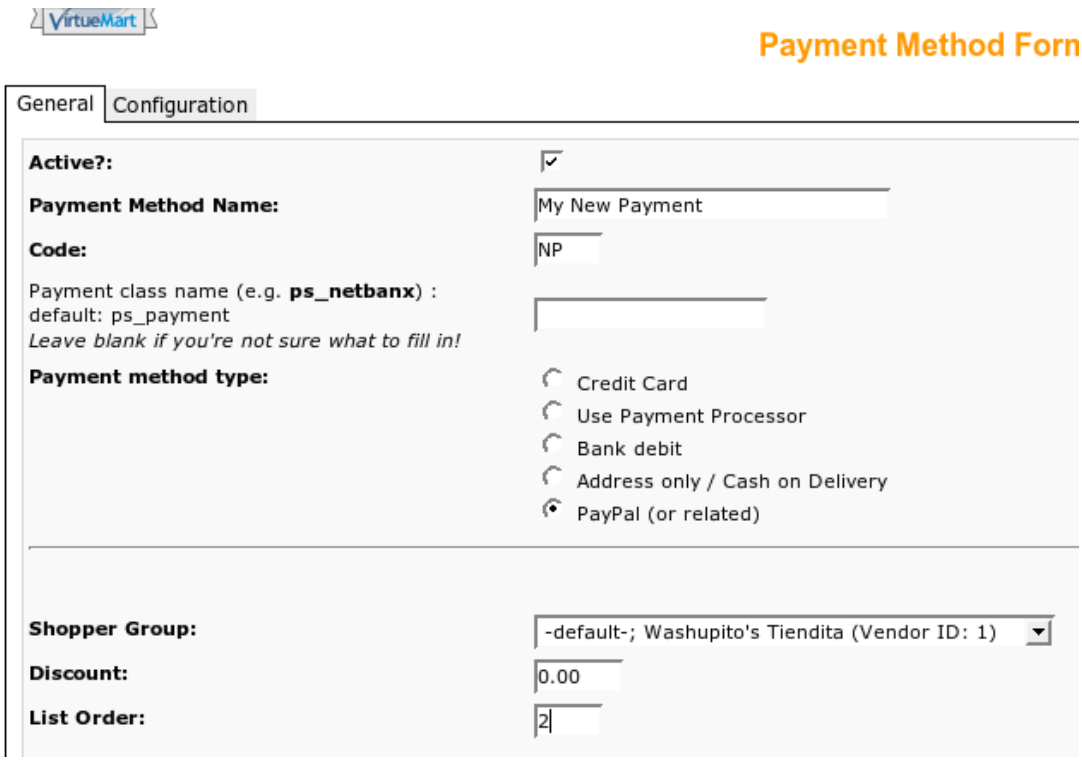

Figure 4.1. Payment Method Form, Tab 1

Note

Be sure that you have NOT selected "credit cart payment" or "automatic processor".

On the second tab you must fill your form code (you can use HTML and PHP!) into the text area called "Payment Extra Info":

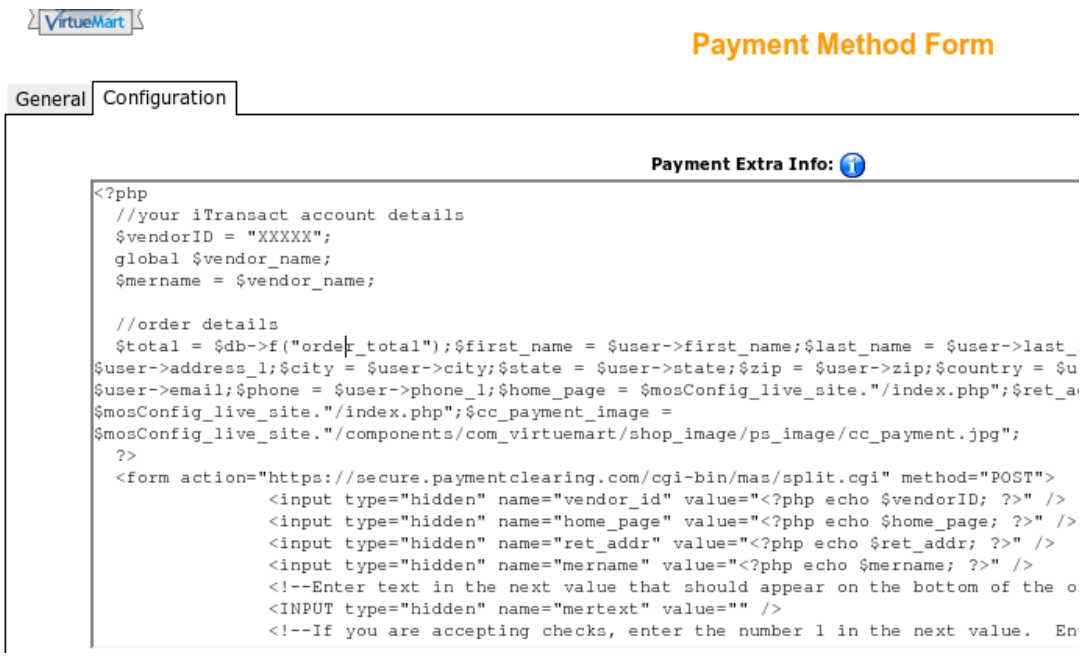

Figure 4.2. Payment Method Form, Tab 2

Caution

The code inside this form MUST BE VALID! If you use PHP code, check if you have written correct code that can be parsed!

4.1.3. Implementation Details

Important

The information in this section only applies to situations when you want to implement a new payment method of the type “Payment Processor” or “HTML-form-based” (simliar to PayPal). All other payment methods can be created by just adding a new payment method in the shop administration! Then you don’t need to create a new payment module.

A payment module implements a technique to either

- communicate with a remote server to authenticate the credit card of a user or
- create a HTML Form to send the user to the pages of the payment provider where he/she can pay and return to your shop afterwards.

All payment modules are located in the directory

`/administrator/components/com_virtuemart/classes/payment/`

and provide two files: the class file and the configuration file.

Example: **Module "mynewpayment"**

You must have two files called

- `ps_mynewpayment.php` (including the class ps_mynewpayment)
- `ps_mynewpayment.cfg.php` (containing all necessary configuration constant definitions)

If the user has chosen to pay using a payment method, which has this class as its processor (entry under *Class Name*), the file `ps_mynewpayment.php` will be included on checkout and its functions will be used to process the payment details, regardless of the implementation.

4.1.4. The API specification

The following is a list of all methods that must be implemented in a payment module's class file.

string **show_configuration**(void)	Shows the configuration form for this payment module in the payment method form.
boolean **has_configuration**(void)	returns true if the payment module can be configured, false if not
boolean **configfile_writeable**(void)	returns true if the configuration file for that payment module is writeable, false if not
boolean **configfile_readable**(void)	

	returns true if the configuration file for that payment module is readable, false if not
void **write_configuration**(Array)	Stores all configuration values for this payment module in the configuration file.
boolean **process_payment**(String $order_number, Float $order_total, Array &$d)	This is the main function for all payment modules that use direct connections to a payment gateway (like authorize.net or eWay XML). This is the place, where the payment details are validated and captured on success. Returns true on sucess, false on failure.
float **get_payment_rate**(Float $subtotal)	This is the function to calculate the fee / discount for this special payment module (so you can calculate a fee, depending on the order total amount).

Note

IF you are about to change configuration variables: do this in both functions: show_configuration and write_configuration!

4.1.5. Installing a Payment Module

Since there's no real installer for payment modules, you must copy the two files `ps_mynewpayment.php` and `ps_mynewpayment.cfg.php` into the directory `/administrator/components/com_virtuemart/classes/payment/` first.

After you have done that, you can add a new payment method ("Store" # "Add Payment Method"). It's important to fill in the correct name for Payment Class Name (in this case: **ps_mynewpayment**) - here's the reason why you must give the class file the same name as the class inside the file: the Shop now tries to include a file called "ps_mynewpayment.php" on Saving the payment method.

When you now re-open the newly created payment method, you have access to the configuration form.

4.2. Shipping Modules

4.2.1. The Basics

Shipping modules are located in the directory

`/administrator/components/com_virtuemart/classes/shipping/`

and have three files: the class file, the information file and the configuration file.

Example: **Module "myShipping"**

You must have three files, called

- `myShipping.php` (including the class myShipping)
- `myShipping.ini` (containing the Name of the Module & the Author and the File Version..)

- `myShipping.cfg.php` (containing all necessary configuration constant definitions)

When activated in the Shop configuration, this payment module will be loaded on the shipping method selection screen, beside all other activated shipping modules.

The shipping rate, a user has selected during checkout is passed from step to step by the parameter **shipping_rate_id**.

This parameter follows a strcit syntax and must be a string build like this:

ShippingClassName|carrier_name|rate_name|totalshippingcosts|rate_id

For our example the shipping rate id for one rate could be:

myShipping|My Carrier|My Rate Name|45.00

The last field (rate_id) can be left empty. The shipping_rate_id parameter is always passed as an urlencoded string.

4.2.2. The Shipping API specification

The following is a list of all methods that must be implemented by a shipping module's class file.

string **list_rates**(Array $d)

Lists all available shipping rates.

> **Tip**
>
> The array `$d` contains the values for the cart total weight (`$d['weight']`) and the ID for the shipping address the user has selected (`$d['ship_to_info_id']`). The ship_to_info_id refers to the field user_info_id in the tables mos_users OR mos_vm_user_info. Check both for a matching entry!

float **get_rate**(Array $d)

Returns the amount for the selected shipping rate by analyzing the parameter shipping_rate_id.

float **get_tax_rate**(Array $d)

Returns the tax rate for this shipping module (e.g. **0.16**).

boolean **validate**(Array $d)

Validates the value for the parameter `shipping_rate_id` usually using isset(`$_SESSION[$shipping_rate_id]`).

Assumes you have set the value in the function list_rates for each returned shipping rate.

void **write_configuration**(Array)

Stores all configuration values for this shipping module in the configuration file.

string **show_configuration**(void)

Shows the configuration form for this shipping module in the shipping module form.

boolean
configfile_writeable(void)

returns true if the configuration file for that module is writeable, false if not

Note

Please always change configuration variables in both functions: show_configuration and write_configuration!

4.2.3. Installing a Shipping Module

Shipping modules also can't be automatically installed, but you must copy the three files mentioned above into the directory

`/administrator/components/com_virtuemart/classes/shipping/`.

After having done that, you must go to the Shop Configuration, where your new shipping module will be automatically recognized (by reading its ini - File) and presented to you as an additional shipping method under the Tab "Shipping".

You can now select it and save the Configuration.

Chapter 5. Developer Guidelines

5.1. General

5.1.1. Using and updating the Changelog

The file CHANGELOG.txt contains the Changelog for the recent Major version of VirtueMart. For every change you make to the source code you must make an entry to that Changelog.

Please use the date, your dev.virtuemart.net username and the following characters to indicate a modification:

```
# -> Bug Fix
+ -> Addition
! -> Change
- -> Removed
! -> Note
```

An example entry could like like this:

```
06-09-2005 soeren
^ changed this and that
# Task #75 [Bug desription]
+ added feature from request Task #56 [feature description]
```

Note

Please keep your descriptions as readable as possible. A lot of people are following the changes and are interested in understanding all changes and their consequences.

Important

If you had to make a change to the database schema, please indicate those changes with extra emphasis. Because you're not the only one working on a development version, please add all queries which are necessary to update a database to comply with your changes.

```
^ ## Database structure changed ##
    ALTER TABLE jos_vm_order_user_info ADD `extra_field_1` varchar(255) defa
```

Please read the section „Database“ for all notes about the database and its scheme.

5.1.2. Compatibility

5.1.2.1. PHP version compatibility

All PHP code written must be compatible down to **PHP version 4.3.0**.

5.1.2.2. MySQL version compatibility

As there is no „real“ database abstraction in Mambo, we keep compatibility to MySQL.

All SQL queries must be compatible with at least **MySQL version 4.0**.

5.1.2.3. Mambo version compatibility

Future versions of VirtueMart will support Mambo versions from 4.5.1a until 4.5.3. It's allowed to copy functionality from a later Mambo version into VirtueMart's ps_main file to maintain compatibility. Compatibility to Mambo 4.5. 1.0.x is not supported.

On the other hand, it is necessary to stay up-to-date with Joomla!. Mambo and Joomla will be developed in two different directions. This process will someday lead to the effect that components written for Joomla, won't work on a Mambo 4.5.3 (or higher).

Note

VirtueMart will keep track with the Joomla development

5.1.3. Accessibility

5.1.3.1. XHTML Compliance

All HTML code used in files must be XHTML compliant. This means syntax like "`<br />`" instead of "`<br>`" and using quotes for attribute values: ** ** instead of ** ** and using lower-case tag- and attribute names: .. instead of <DIV>...</DIV>.

5.1.3.2. Javascript

Javascript can be used in the frontend (is NO problem at all in the backend and for all administration pages).

But in the frontend all functionality that is used by a customer must also work with Javascript disabled! This includes Javascript-based category trees (always also include a <noscript>Non JS code</noscript> section for people who have disabled Javascript).

5.2. Coding Guidelines

Register Globals is Off

All code must work with PHP register_globals = Off.

PHP Code Tags

Always use `<?php ?>` to delimit PHP code, not the `<? ?>` shorthand. This is required for PEAR compliance and is also the most portable way to include PHP code on differing operating systems and setups.

Single Quotes vs. Double Quotes

- Use single quotes to refer to an index between brackets of an array (ex: $foo['name'] and not $foo[name] or $foo["name"])
- Use single quotes instead of double quotes as much as possible because it's faster to parse.

Line Spacing

Indent using 4 spaces or a tab.

Order and Spacing

To ease long term readability of source code, the text and tags must conform to the order and spacing provided in the example above. This standard is adopted from the JavaDoc standard.

SVN ident strings

Include SVN $Header$ strings in your code. This makes it easier for people to know which version of a file they have and where it came from, so that they can usefully refer to the file's SVN history to find out about bugs and fixes, etc. If your repository is configured appropriately, use the custom tag instead of $Header$.

Variable Settings

- Always run Mambo/Joomla! and PHP will full Error Reporting Level (E_ALL). You can change this level in the global configuration (see „Server“ # Error Reporting Level) and in your PHP.ini.
- Always initialize variables. (just $a=0 is initialization)
- Use isset($var) to check if a variable has been set. Use empty($var) to check if Array indexes have been set or are empty.

Header Comment Blocks

All source code files in the repository shall contain a "page-level" docblock at the top of each file and a "class-level" docblock immediately above each class. Below are examples of such docblocks.

```
<?php

/**
 * Short description for file
 *
 * Long description for file (if any)...
 *
 *
 * @package    VirtueMart
 * @subpackage classes_product
 * @author     Original Author <author@example.com>
 * @author     Another Author <another@example.com>
 * @copyright  2007 VirtueMart Developer Team
 * @license    http://www.gnu.org/copyleft/gpl.html GNU/GPL
 * @version    $Id: Developer_Manual.xml 1079 2007-12-07 18:34:23Z soeren_nb
 */

/*
 * Place includes, constant defines and $_GLOBAL settings here.
 * Make sure they have appropriate docblocks to avoid phpDocumentor
 * construing they are documented by the page-level docblock.
 */

/**
 * Short description for class
 *
 * Long description for class (if any)...
 *
 * @author     Original Author <author@example.com>
 * @author     Another Author <another@example.com>
```

```
 * @copyright  2004-2007 VirtueMart Developer Team
 * @license    http://www.gnu.org/copyleft/gpl.html GNU/GPL
 * @version    Release:
 */
class foo {
    /** @var database Internal database class pointer */
    var $_db=null;
    /** @var object An object of configuration variables */
    var $_config=null;
    /** @var object An object of path variables */
    var $_path=null;
    /** @var mosSession The current session */
    var $_session=null;
    /** @var string The current template */
    var $_template=null;
    /** @var array An array to hold global user state within a session */

    /**
    * This function does something special.
    * @since VirtueMart 1.0.1
    * @param string The name of the product
    * @param int The ID of the product
    * @return string HTML Table with a "snapshot" of the product
    */
    function myFunction( $arg1, &$arg2 ) {

    }
}

?>
```

Required Tags That Have Variable Content

Short Descriptions

Short descriptions must be provided for all docblocks. They should be a quick sentence, not the name of the item, but the description of the „what does this file / class?“.

@license

VirtueMart is released under the GNU/GPL license. You should keep this license for best compatibility.

```
* @license    http://www.gnu.org/copyleft/gpl.html GNU/GPL
```

@author

There's no hard rule to determine when a new code contributor should be added to the list of authors for a given source file. In general, their changes should fall into the "substantial" category (meaning somewhere around 10% to 20% of code changes). Exceptions could be made for rewriting functions or contributing new logic.

Simple code reorganization or bug fixes would not justify the addition of a new individual to the list of authors.

Optional Tags

@copyright

Feel free to apply whatever copyrights you desire. When formatting this tag, the year should be in four digit format and if a span of years is involved, use a hyphen between the earliest and latest year. The copyright holder can be you, a list of people, a company, the PHP Group, etc. Examples:

```
* @copyright 2003 John Doe and Jennifer Buck
* @copyright 2001-2004 John Doe
* @copyright 2005 XYZ Corporation
```

5.3. SVN Access

5.3.1. How to obtain the latest VirtueMart source code from SVN

5.3.1.1. General Information

In order to download source code from the SVN repositories you'll need a SVN client software. A recommended SVN client is SmartSVN [http://www.syntevo.com/smartsvn/] or TortoiseSVN [http://tortoisesvn.tigris.org/].

You can access the repository read-only anonymously by using an empty password.

Please note that the SVN server is case-sensitive. Fill in the details into your SVN client just as they are provided here.

5.3.1.2. Development Version

You can checkout the development version of VirtueMart from https://dev.virtuemart.net/svn/virtuemart/trunk.

If the "trunk" is downloaded, you will have 4 new directories:

- build_scripts (batch/shell scripts to build installable archives)
- documentation (the documentation sources)
- languagemanager (the language manager component to modify language files and add/modify/delete language tokens)
- virtuemart (the VirtueMart 1.1.x sources)

5.3.1.3. Latest code from VirtueMart 1.0.x (stable branch)

You can checkout the latest version from the stable branch of VirtueMart from https://dev.virtuemart.net/svn/virtuemart/branches/virtuemart-1_0_0.

If the branch "virtuemart-1_0_0" is downloaded, you will have 2 new directories:

- build_scripts (batch/shell scripts to build installable archives)
- virtuemart (the VirtueMart 1.0.x sources)

5.3.1.4. How to build an installable archive from the sources

Before you can install VirtueMart you will have to build installable archives. There are 2 types of build scripts available - one version for Windows and one for Linux, which can help you build the correct archives. You should have got the build scripts by following the instructions from the first section on this page.

Important

Before you run the scripts make sure that 7Zip is installed on your computer.

1. Configure the scripts!

   First you must open the files and change the paths inside to match your configuration.

2. Build your own packages by running the script:

   - Build_Component.bat - (Win) build just the component archive
   - Build_VirtueMart_complete.bat - (Win) build the whole “Complete Package” including modules and mambots
   - build_component.sh - (Linux) build the component archive
   - build_virtuemart_complete.sh - (Linux) build the whole “Complete Package” including modules and mambots

5.3.2. Documentation Sources

The VirtueMart Project manages the documentation in the DocBook [http://docbook.org] format. You can checkout the sources in the DocBook format and transform the DocBook source using an XSL Transformer into PDF, HTML, CHM or whatever else... All you have to do is checkout the directory `/trunk/documentation` from dev.virtuemart.net (details see above).

The easiest and most professional software to modify DocBook sources is the XML Mind XML Editor [http://xmlmind.com/xmleditor/download.shtml]. After you have downloaded and installed the software you can just open the main documentation file and start working on it:

- /trunk/documentation/User_Manual/User_Manual.xml (the User Manual)
- /trunk/documentation/Developer_Manual/Developer_Manual.xml (the Developer Manual)

5.4. Using SVN

5.4.1. Basic points

This section describes things that are generally applicable when using SVN; guidelines that are more specific to particular tasks or uses are described in the other sections.

5.4.1.1. When to check in

Check in early, check in often. When you have made a change that works, check it in. Check in separate changes in separate commits (as much as possible). Don't be shy to check in work-in-progress, so long as it is minimally functional, or at least compilable without errors.

5.4.1.2. Commit messages

Use meaningful commit messages. Explain what bug the commit fixes, or what features it adds. Don't be too concise: "fixed typo" is too short; "fixed Task# 2432" or "fixed typo in function name" is OK. The aim is to make it easier to find the desired change easily from just the commit messages.

The converse of this is including too much information. SVN automatically maintains information like the date and time of the commit, who made the commit, what code was changed, etc. You don't need to include it in the commit message yourself.

5.4.1.3. Using tags

If in doubt, lay down a tag. Tags are useful for pinning down a particular version of the code, e.g. one that is being run in service, or just before a big change or import. They are also used to identify branches. Tag names should be short and meaningful, like variable names. For example, `virtuemart-20051222`, `pre-new-virtuemart`, `fanf-patches`, corresponding to the uses mentioned above. Tags should be commented in the modules file.

5.4.2. Code

Most of the guidelines in this section are common sense, but it's worth while re-iterating them in the context of SVN because it has implications that might not be immediately obvious.

5.4.2.1. Never reformat code

Never, ever reformat code. This is a really bad thing to do because it makes diffs hard to understand and apply. Upstream authors won't accept patches against reformatted code. Bugfixes and patches against the upstream code won't apply. New versions of the upstream code can't be imported. Real changes get hidden in the mass of reformatting.

No-one's favourite coding style is significantly better or worse than anyone else's so reformatting code provides no advantage to oppose the disadvantages.

5.4.2.2. Format code consistently

Use the same coding style as the code you are editing. This is a corollary to the previous subsection. It is easier for people reading the code if it uses consistent layout rules throughout, so when you are editing someone else's code the code you add should be in the same style.

5.4.2.3. Tab settings

Tabs are four characters wide. This is also a corollary to the previous subsections. Although indentation sizes vary greatly, tabs are almost universally eight characters, so using a different setting is liable to cause confusion or even reformatting. A four character tab might suit your indentation style, but the rest of the world will think your code is a mess.

5.4.2.4. Comments

Commit messages are not a substitute for comments, or vice versa. Comments should describe data structures and why the code does what it does; commit messages should explain why code was changed.

5.4.2.5. SVN ident strings

Include SVN `$Header$` strings in your code. This makes it easier for people to know which version of a file they have and where it came from, so that they can usefully refer to the file's SVN history to find out about bugs and fixes, etc.

If your repository is configured appropriately, use the custom tag instead of `$Header$`.

5.4.3. Handling tricky situations

Because of limitations in SVN certain tasks are inherently difficult, particularly recovering from mistakes. Although changing the repository directly is nearly always a *Really Bad Idea* sometimes it cannot be avoided. These guidelines explain what to do in these situations.

5.4.3.1. "Whoops! I checked in the wrong thing!"

Once a change has been committed you cannot un-commit it. You have to reverse the change and check in a new revision with the old code.

Sometimes you might have a number of changes in your working copy which should be committed separately but accidentally get committed all at once with a commit message that's only appropriate to one of the changes. The safe thing to do is revert the inadvertent commits then re-commit them with the right message; editing the repository directly is possible but foolishly dangerous.

5.4.3.2. "Whoops! I cocked up a `svn import`!"

Getting an import right is important because it affects the long-term usefulness of the repository. Check import commands particularly carefully before running them!

If you do make a mistake, the solution depends on exactly what went wrong. You might have run the command in the wrong working directory, or you might have used the wrong repository path, etc. The important point is whether the imported files coincide with files in the repository or not.

1. If none of the files in the erroneous import have the same name as an existing file in the repository (e.g. they all ended up in a completely new directory) then just removing the files from the repository can be done by using the appropriate `rm` command in the repository.

2. If the import is OK apart from an incorrect tag, the tag can probably be deleted and re-applied correctly without too much pain. (This may not be true for a misspelled vendor branch tag.)

3. If there is a filename clash with an unrelated file, then there's a fairly serious problem. Find a SVN guru and help him or her to fix the repository manually. You won't be popular.

5.5. Database

Changes to the main db schema **require** a that an upgrade patch is posted as well. Your change will be backed out if you don't provide a patch as well.

5.5.1. Changelog!

First of all you need to make an entry in the Changelog, including the SQL Queries to update a database scheme.

5.5.2. SQL Update File

All changes to the database scheme are collected in an SQL file. There's a file for each minor version jump, e.g.

```
UPDATE-SCRIPT_com_virtuemart_1.0.x-to-1.1.0.sql
```

The file can be found in the subdirectory /sql.

A user must see which version of VirtueMart this SQL patch file applies to and to which version it updates the db scheme.

In this case the SQL file would update a db scheme from version

VirtueMart 1.0.x to **1.1.0**

Chapter 6. About the Project

6.1. The Project

VirtueMart is an Open Source E-Commerce solution for Joomla! and Mambo.

VirtueMart is released under the GNU Public License. It is free to download and use.

There's a small group of developers who help in making this Shopping Cart Script more professional and easily usable. The author of this documentation is the Lead Developer...and having not much time besides Wife, Work & Studies.

History: VirtueMart is the successor of mambo-phpShop. mambo-phpShop was the first port of phpShop to Mambo. phpShop was developed by Edikon Corp. [http://www.edikon.com] and the phpShop community - http://www.phpshop.org.

6.2. Documentation

This documentation was written using XMLMind XML Editor [http://www.xmlmind.com/xmleditor] using the DocBook [http://www.docbook.org] XML Format.

DocBook defines a set of markup elements useful for marking up text so that the text can then be transformed into several different formats. It's possible to create documents in different formats: PDF, HTML, HTML Help (.chm Files for Windows Help), XML, RTF, TeX, WordML (Word 2003) and others. The author of this document uses eDE [http://docbook.e-novative.de/] for generating the End-User documents. The idea is to write just once and reach the largest possible number of people with the information. Digital information not stored properly tends to get lost. Due to the fact that not containing uncommon characters (such as binary formats) it's possible to index and search directly on the documents written on SGML and consequently on DocBook. The SGML systems use markups to make their description. DocBook holds over 300 markup elements each one with several attributes which can assume several values, these can be fixed or defined by the document / style that the author has used.

6.3. Homepage, Forum, Developer Resources

The project homepage is http://virtuemart.net.

The Forum can be found at http://forum.virtuemart.net. You are invited to join Discussions at our Developer Board!

The Central Developer Platform for VirtueMart and Open Source Projects related to VirtueMart can be found at https://dev.virtuemart.net. That's the place where VirtueMart can be downloaded. And you can create your own Project there! We offer free Project Hosting for your VirtueMart-related Open Source Project.