VirtueMart Developer Manual

Soeren Eberhardt(www.virtuemart.net [http://www.virtuemart.net])

Revision History

Revision 1.1	November 21, 2005	soeren_nb
	Update for VirtueMart	
Revision 1.0	June 29, 2005	soeren_nb
	Initial Release (mambo-phpShop v1.2 stable-pl3)	

Table of Contents

1. Preambel

The content of this document is related to VirtueMart.

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

Conversion to Joomla and much more: © 2005 Sören Eberhardt

The Software 'VirtueMart' is intended for use in Joomla and Mambo (version 4.5.1 and 4.5.2.x). 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.*

2. Introduction

2.1. History

VM has its roots in a Shop 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 Joomla-phpShop until September 2005. In October 2005it was renamed to VirtueMart.

2.2. Differences to phpShop

VM still contains some code parts from phpShop, but experiences phpShop coders will see 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offset	Outdated/removed The offset parameter was completely replaced by the parameter "limitstart", which is Joomla standard for page navigation.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.

2.3. 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).

3. Basics

3.1. 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.phpshop.html.php` (controls the administrative Toolbar)

/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 presentation of HTML Code.

They are ordered by shop 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/html/templates/

Contains Templates for some pages

../basket	Templates for Cart Display.
../browse	Templates for Product Listing Pages (can be assigned in the Category Form)
../order_emails	Templates for the Order Confirmation Email
../product_details	Templates for the Product Details Pages.

/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 administratin (`admin.css`)
/components/com_virtuemart/js/	Contains Javascripts (WebFX – Tabs, JSCookTree and the IE-PNG-Transpareny Fix)
/components/com_virtuemart/shop_image/	

/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

3.2. Main Flow Chart

3.2.1. 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.phpshop.php` in it.

If found, the file is included.

3.2.2. 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`.

3.3. Core Modules & their Functions, Environment Variables

3.3.1. 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.

3.3.2. 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 `mos_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.$db->f("function_class").".php" );

// create an object
$string = "\$" . $func_class . " = new " . $func_class . ";";
eval( $string );

// RUN THE FUNCTION
$cmd = "\$ok = \$" . $func_class . "->" . $func_method . "(\$vars);";
eval( $cmd );
```

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.

3.3.3. 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.

3.3.4. 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 a f
  ```

-
  ```
  $vmLogger->warning( 'My warning message to the user' ); // Mainly used to trigger war
  ```

-
  ```
  $vmLogger->notice( 'My Notice to the user' );
  ```

-
  ```
  $vmLogger->info( 'My informational message to the user' ); // Used to give success me
  ```

-
  ```
  $vmLogger->debug( 'My debug message to the user' ); // Only displayed when DEBUG is e
  ```

-
  ```
  $vmLogger->tip( 'My advice to the user' ); // Used to display Advice messages to the
  ```

3.4. 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.

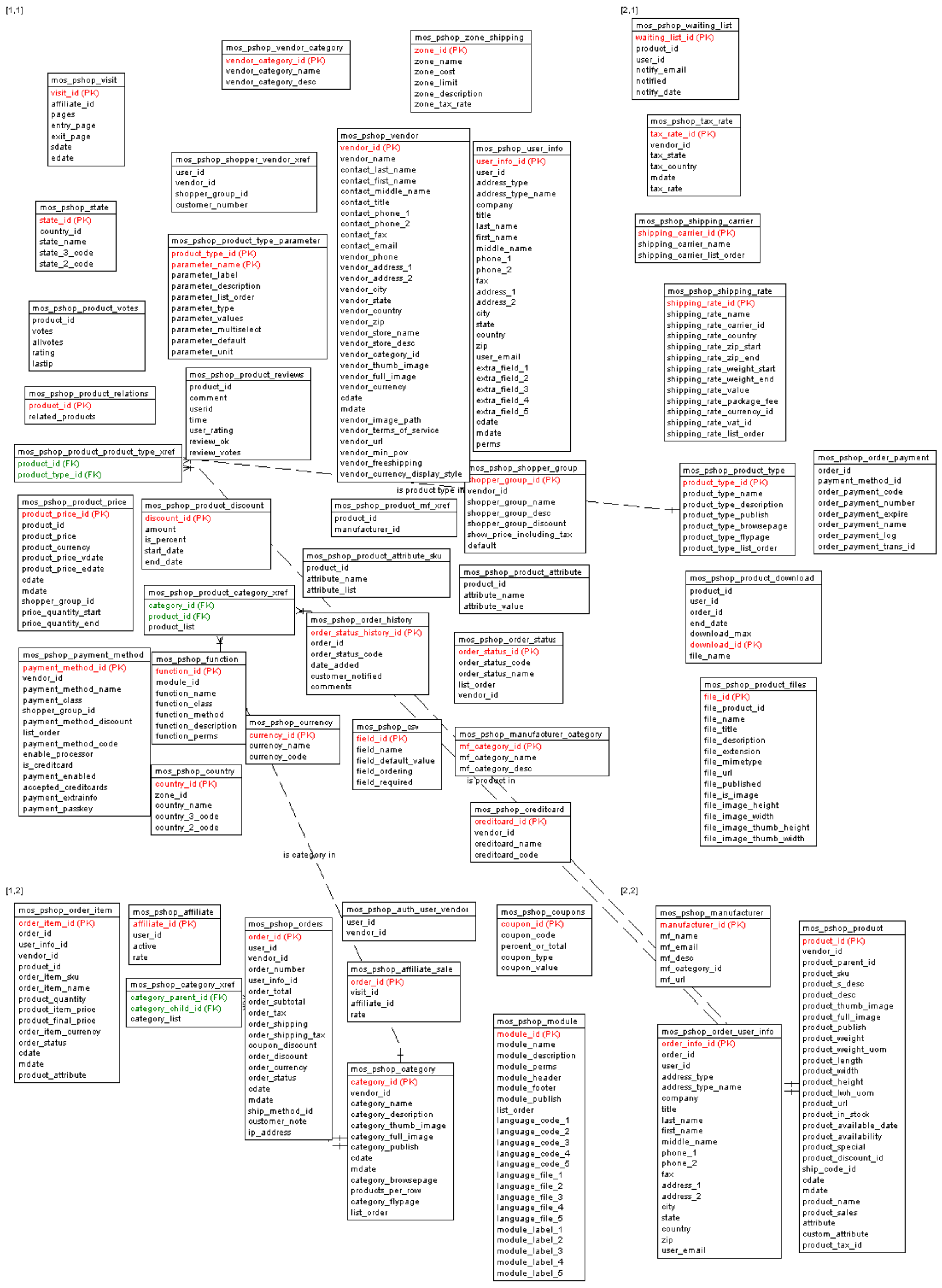


Figure 1. Entitiy Relationship Diagram

3.5. 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! }
  ```

3.6. 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

4. 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)!

4.1. 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))

4.2. Modifying Templates

VM doesn't use a template engine (like patTemplate or Smarty) to parse its templates.

4.2.1. Flypage Templates

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

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

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`.

The following placeholders are used:

./.

4.2.2. 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 `/html/templates/browse/`.

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

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`.

The following placeholders are used:

./.

4.2.3. 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/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`.

The following placeholders are used:

./.

4.2.4. Basket Templates

Basket templates control the layout of the basket.

The templates can be found in the directory `/html/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 concept in the basket templates is another one than in the other template files, because loops are used. So we have a minimum number of PHP statements, which can be easily understood by designers without much PHP knowledge.

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

5. 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.

5.1. Payment Modules

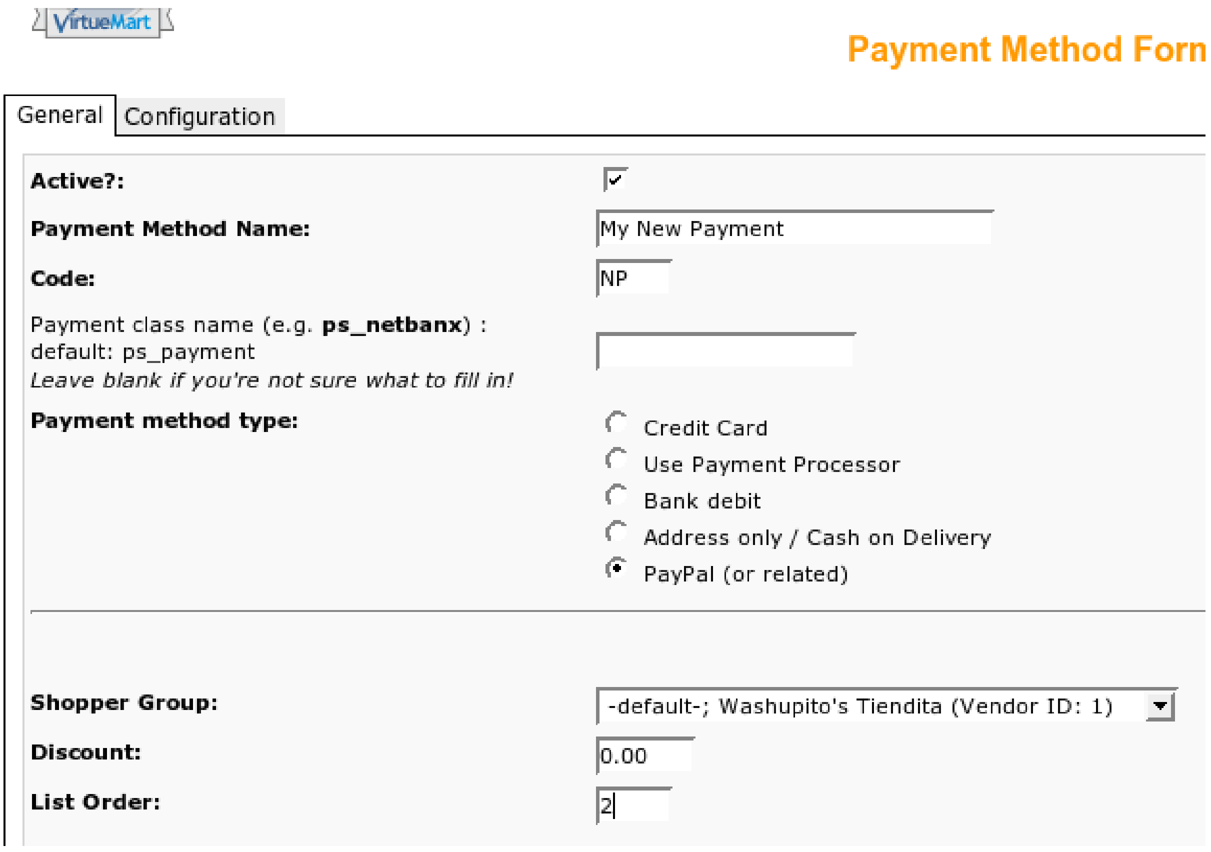
There are two general types of payment modules in VM:

- automatic credit card processors which do server-to-server communication before the order is placed (e.g. authorize.net AIM)
- all other payment methods that do not communicate to a server before the order is placed (PayPal, World-pay, all other form-based payment methods, all formless payment methods)

5.1.1. 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:

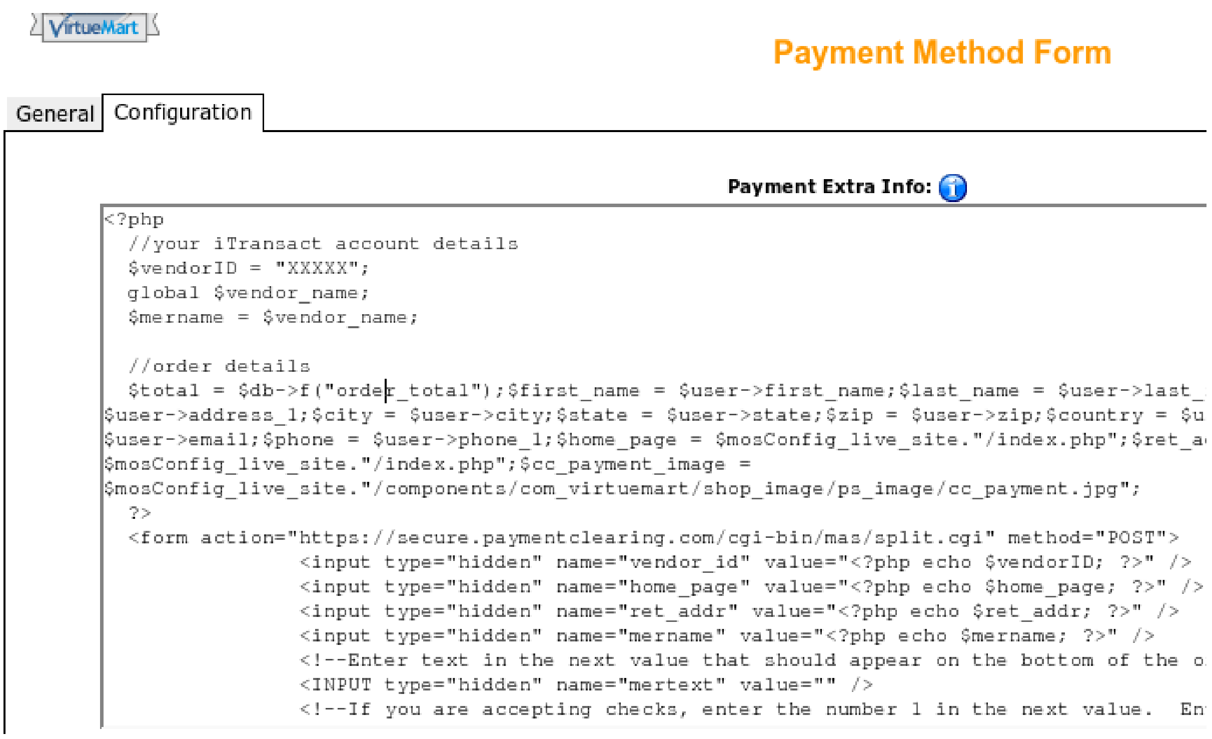


Active?:	✓
Payment Method Name:	My New Payment
Code:	NP
Payment class name (e.g. **ps_netbanx**) : default: ps_payment *Leave blank if you're not sure what to fill in!*	
Payment method type:	Credit Card Use Payment Processor Bank debit Address only / Cash on Delivery [✓ PayPal (or related)]
Shopper Group:	-default-; Washupito's Tiendita (Vendor ID: 1)
Discount:	0.00
List Order:	2

Note

Be sure that you have NOTselected "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":



```
<?php
  //your iTransact account details
  $vendorID = "XXXXX";
  global $vendor_name;
  $mername = $vendor_name;

  //order details
  $total = $db->f("order_total");$first_name = $user->first_name;$last_name = $user->last_
$user->address_1;$city = $user->city;$state = $user->state;$zip = $user->zip;$country = $u
$user->email;$phone = $user->phone_1;$home_page = $mosConfig_live_site."/index.php";$ret_a
$mosConfig_live_site."/index.php";$cc_payment_image =
$mosConfig_live_site."/components/com_virtuemart/shop_image/ps_image/cc_payment.jpg";
  ?>
  <form action="https://secure.paymentclearing.com/cgi-bin/mas/split.cgi" method="POST">
              <input type="hidden" name="vendor_id" value="<?php echo $vendorID; ?>" />
              <input type="hidden" name="home_page" value="<?php echo $home_page; ?>" />
              <input type="hidden" name="ret_addr" value="<?php echo $ret_addr; ?>" />
              <input type="hidden" name="mername" value="<?php echo $mername; ?>" />
              <!--Enter text in the next value that should appear on the bottom of the o
              <INPUT type="hidden" name="mertext" value="" />
              <!--If you are accepting checks, enter the number 1 in the next value.  En
```

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!

5.1.2. Digging deeper: The Basics

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.

5.1.3. 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!

5.1.4. 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.

5.2. Shipping Modules

5.2.1. 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.

5.2.2. 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!

5.2.3. 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.

6. About the Project

6.1. CVS Access

6.1.1. VirtueMart Source Code

This project has its CVS repository on the sourceforge.net CVS Server. You can checkout the module `VirtueMart` from **cvs.sourceforge.net**. In order to obtain the source anonymously (read only) you need to know the following:

Connection Type: **pserver**
CVS Server: **cvs.sourceforge.net**
CVSROOT: **/cvsroot/virtuemart**
Module Name: **virtuemart**
User: **anonymous** (no password required)

Warning

The CVS server is case-sensitive. Fill in the details in your CVS Program (e.g. www.tortoisecvs.org [http://www.tortoisecvs.org] for Windows) just as they are provided here.

6.1.2. Documentation Sources

The VirtueMart Project manages its documentation in the DocBook 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 module documentation from cvs.sourceforge.net.

Connection Type: **pserver**
CVS Server: **cvs.sourceforge.net**
CVSROOT: **/cvsroot/virtuemart**
Module Name: **documentation**
User: **anonymous** (no password required)

Warning

The CVS server is case-sensitive. Fill in the details in your CVS Program (e.g. www.tortoisecvs.org [http://www.tortoisecvs.org] for Windows) just as they are provided here.

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

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

There we also have a forum and you are invited to join our developer board!