WHMCS Provisioning Module Docs

Last Updated: 8th July 2011

Contents

Introduction

Provisioning Modules, also referred to as Product or Server Module, allow you to create modules to allow for the provisioning and management of products & services.

The core functionality of a module is for creating, suspending, unsuspending & terminating of products as various events occur such as new orders being paid for, items becoming overdue, overdue invoices being paid & cancellations being requested.

However a module in WHMCS can do much more than that, including automated password resets, upgrades/downgrades, renewals, admin based links, client area output, and more via custom functions.

Other types of modules that can be created in WHMCS are Payment Gateways, Domain Registrars and Admin Addon Modules so if you're looking to create one of those then you need alternative documentation available from

http://docs.whmcs.com/Developer_Resources

So if you're still with us… let's get started!

Getting Started

Naming Conventions

All modules must be given a filename that consists of a single word, and all lowercase letters.

Modules reside at a path of /modules/servers/filename/filename.php

Within the module itself, all functions are prefixed with the module filename, followed by an underscore, and then the function name. For example if your module was called “mymodule” the create function would be named as “mymodule_CreateAccount”.

The ConfigOptions Function

The only required function of every provisioning module is the ConfigOptions function. This is what defines the options that can be set on a per product basis when assigning a product to this module. This is often used for things like Package Names, Resource Limits, etc…

The supported field types are:

- Text
- Dropdown
- Password
- Yesno

Text and password fields can be given a “Size” parameter to define the length of the field, while dropdown fields require a comma separated list of options in the “Options” parameter.

The sample module template included in the dev kit includes an example of defining each of the supported field types.

Note: There is a limit of a maximum of 24 config options for a module.

Supported Functions

What follows here is a brief overview of all the possible functions that a product provisioning module in WHMCS can contain. All functions within a module are optional and can be omitted from the module if they don't apply. Within the code itself, all functions are prefixed with the module filename, followed by an underscore, and then the function name as shown in bold in these descriptions.

CreateAccount

This function is called when a new product is due to be provisioned. This can be invoked automatically by WHMCS upon checkout or payment for a new order, or manually by an admin user from the Products/Services tab under a clients profile within the admin area.

SuspendAccount

This function is called when a suspension is requested. This is invoked automatically by WHMCS when a product becomes overdue on payment, or can be called manually by admin user.

UnsuspendAccount

This function is called when an unsuspension is requested. This is invoked automatically upon payment of an overdue invoice for a product.

TerminateAccount

This function is called when a termination is requested. This can be invoked automatically for long overdue products if enabled (auto termination is disabled by default in Setup > Automation Settings) or requested manually by an admin user.

Renew

This function is called each time a renewal invoice for a product is paid.

ChangePassword

This function is called when a client requests a password change from the client area. For this option to show up this function must be declared in the module, the status of the product must be active. Admins can also invoke this command from the admin area.

ChangePackage

This function is used for upgrading and downgrading of products. This function will be called automatically when an upgrade or downgrade order placed by the client is paid for, or can be invoked manually by admin users from the product management pages. This same function is called for upgrades and downgrades of both products and configurable options.

ClientArea

This function can be used to define module specific client area output. It accepts a return of HTML for display on the product details page of the client area. The output can alternatively be specified via a template file within the module folder named "clientarea.tpl" to allow for end user customisation. This function is discussed in more detail later on in the docs.

AdminArea

This function is used to define HTML code to be displayed within the admin area server configuration page (Setup > Servers) and is commonly used to provide an automated shortcut/login link to the server control panel being integrated with.

LoginLink

This function can be used to define HTML code used to link specifically to the customers account within a servers control panel. It is displayed on the product management page within the admin area if defined and must be an HTML output or link (no forms).

ClientAreaCustomButtonArray

This function can be used to define custom functions that your module supports that customers are allowed to invoke and run from the client area. These functions can perform actions or product page output in the clientarea. Some example usages for this are to provide product management pages, bandwidth reporting pages, etc…

AdminCustomButtonArray

This function can be used to define custom functions that your module supports, similar to the above, but that admin users are allowed to run. This can contain more functions than the client area is allowed.

UsageUpdate

This function is used to perform a daily import of the disk and bandwidth usage for accounts from a server. The data imported is then used to display the usage stats both within the client and admin areas of WHMCS, and also in disk and bandwidth overage billing calculations if enabled for a product.

AdminServicesTabFields

This function can be used to define additional fields and output to be displayed on the admin product management pages.

AdminServicesTabFieldsSave

This function is used in conjunction with the above to handle the values submitted in any custom fields when a save request is submitted.

Module Parameters

Every module function is passed the same set of parameters. These variables provide information about the specific product/service the module command is being invoked for, along with the settings from the product itself as defined in the product configuration.

Var Name	Description
serviceid	the unique ID of the service Database Field: tblhosting.id
pid	the product ID that the service is assigned to Database Field: tblproducts.id
serverid	the server ID that the service is assigned to Database Field: tblservers.id
domain	the domain entered by the customer when ordering Database Field: tblhosting.domain
username	the username generated for the service (defaults to first 8 letters of the domain) Database Field: tblhosting.username
password	the password generate for the service (10 char randomly generated on first creation consisting of letters & numbers, both upper & lowercase) Database Field: tblhosting.password
producttype	the product type which can be one of sharedhosting, reselleraccount, server or other
moduletype	the module name being called (will match filename of module)
configoptionX	with X being from 1 to 24, these fields contain all the module settings for the product in the order defined in the ConfigOptions function of the module
clientsdetails	This variable contains a sub-array of all the clients details for the client who owns the service in question. This contains things like firstname, lastname, email, address1, country, etc…
customfields	this variable contains a sub-array of all the custom fields defined for the product, with the key being the custom field name - $params[‘customfields’][‘Field Name’]
configoptions	This variable contains a sub-array of all the configurable options defined for the product, again with the key being the option name in this case - $params[‘configoption’][‘Option Name Here’]
server	true/false to define if the product is assigned to a server
serverip	the IP Address of the selected server
serverhostname	the Hostname of the selected server
serverusername	the Username of the selected server
serverpassword	the Password of the selected server
serveraccesshash	the Access Hash of the selected server
serversecure	true/false to define if Use SSL is enabled in the Server Configuration

Config Options

Config options (not to be confused with Configurable Options) are the module settings defined on a per product basis. This are supplied as a numbered list, so the first option would be $params['configoption1'] , the second $params['configoption2'], etc… The order is defined by the order in which you specify the settings in the ConfigOptions function of the module.

Custom Fields & Configurable Options

The values from any custom fields & configurable options are loaded and passed into modules as parameters so that they can be easily used. They are passed as an array with the key being the name of the field or option.

For example if you created a custom field called "Username", then that would be referenced using $params['customfields']['Username']

Similarly if you created a configurable option named "Disk Space", then that would be referenced using $params['configoptions']['Disk Space']

Core Module Functions

The core module functions are the **Create, Suspend, Unsuspend, Terminate, Renew, ChangePassword** and **ChangePackage** functions.

These 7 functions all operate in a very similar manner. They can all be invoked both manually and automatically, and they are all expected to return either a success or error response.

Response Handling

Each one of these functions after performing the actions they are required to, must either return a success message or error response.

For a successful result the code must actually return the word "success" to end the function. If WHMCS receives that it will know that the function completed as intended and continue on that basis.

However, should the function fail, what you return should be a user understandable error message, as it will be displayed directly to staff users.

Actions

When a function is successful, there are various actions that are performed as follows

Function Name	Successful Events
CreateAccount	Changes status to Active + Sends Product Welcome Email
SuspendAccount	Changes status to Suspended
UnsuspendAccount	Changes status to Active
TerminateAccount	Changes status to Terminated
Renew	None
ChangePassword	Updates password in database
ChangePackage	None

In addition to the above actions, admin users are given a confirmation of successful functions completing, and errors in the case of them failing. If the functions were invoked automatically such as on payment of a new order, then that notification can be in the form of an email. And in the case of the ChangePassword function, any errors returned from that function are also displayed to end users within the client area also.

Client Area Output

Another key function of a module is to give the client access to options within the client area. This is done using the ClientArea function of a module.

Function vs Template

The ClientArea function within the modules PHP code can simply return HTML output to be displayed within the product details view on the client side. However, another option for defining the output is to create a template file with the name of "clientarea.tpl" within your module folder which if found will then be loaded and displayed in place of the ClientArea function.

The template file will be called with all the same module parameters as other functions get defined and ready to use as smarty template variables so if you want to make the client area output of your module customisable then using the template method is the perfect way to do it.

Actions/Events

Quite often you might want to have buttons or links in the output of your module that can do various things. One very simple way of doing this is to link back to the product details page that the client is already on with an additional var in the request, and then to check for the existence of that var within your template code. For example:

```
 9
10  <form method="post" action="clientarea.php?action=productdetails">
11  <input type="hidden" name="id" value="{$serviceid}" />
12  <input type="hidden" name="bwstats" value="1" />
13  <input type="submit" value="Bandwidth Statistics" />
14  </form>
15
16  {php}
17
18  if (isset($_POST['bwstats'])) {
19
20      # Bandwidth stats function was called
21
22      echo 'Output here...';
23
24  }
25
26  {/php}
27
```

An alternative solution is using custom functions which we will look at now.

Custom Functions

Custom functions allow you to define additional operations that can be performed using the module. The custom functions can do anything you want, and return either a success or failed response. Permission can then be granted for who can use each custom function, be it just clients, just admins, or both.

The naming convention for custom functions follows the same as any other function within a module so it must begin with the module filename, followed by an underscore, and then the custom function name.

The easiest way to demonstrate this is with an example so let's take an example of a reboot & shutdown function in the case of a module for a VPS system:

```
139
140 function template_reboot($params) {
141
142     # Code to perform reboot action goes here...
143
144     if ($successful) {
145         $result = "success";
146     } else {
147         $result = "Error Message Goes Here...";
148     }
149     return $result;
150
151 }
152
153 function template_shutdown($params) {
154
155     # Code to perform shutdown action goes here...
156
157     if ($successful) {
158         $result = "success";
159     } else {
160         $result = "Error Message Goes Here...";
161     }
162     return $result;
163
164 }
165
```

In the above you can see how the custom functions are defined, being passed all the same module parameters as all other functions, & returning either "success" for a success or an error message to indicate a failure. Now let's say we wanted to allow clients to perform reboots, but only admin users to be able to perform a shutdown, here's how we would define that:

```php
function template_ClientAreaCustomButtonArray() {
    $buttonarray = array(
     "Reboot Server" => "reboot",
    );
    return $buttonarray;
}

function template_AdminCustomButtonArray() {
    $buttonarray = array(
     "Reboot Server" => "reboot",
     "Shutdown Server" => "shutdown",
    );
    return $buttonarray;
}
```

What we are saying here is that clients are allowed perform the “reboot” function, but admins can perform both “reboot” and “shutdown”.

The text that’s the key part of the array is what’s displayed on the button in the row of module command buttons to an admin user, and the value is the custom function name excluding the modulename_ prefix.

So now all that’s left is to give the client the option to perform the custom function from the client area, and for that we go back to the previous section on Client Area Output. You must output a button linking to invoke the custom module for this and an example of how to do this is as follows:

```php
<form method="post" action="clientarea.php?action=productdetails">
<input type="hidden" name="id" value="{$serviceid}" />
<input type="hidden" name="modop" value="custom" />
<input type="hidden" name="a" value="reboot" />
<input type="submit" value="Reboot VPS Server" />
</form>
```

Where “a” in the above form post is the custom module function to run.

Additional Client Area Pages

A feature which is available from WHMCS V4.2 and later is the ability to create custom client area pages from within a module.

This is done using the custom function methods above, but instead of performing a function and simply returning success or an error, you instead return an array defining the parameters for the page.

The array has to contain a template filename to display, a breadcrumb path showing the end user where they are, and any additional template variables that you want to define such as the results of API calls to be used within the custom template.

The custom template name you define is read from within the module folder (not the active client area template folder).

An example of calling a template file named “example.tpl” is below:

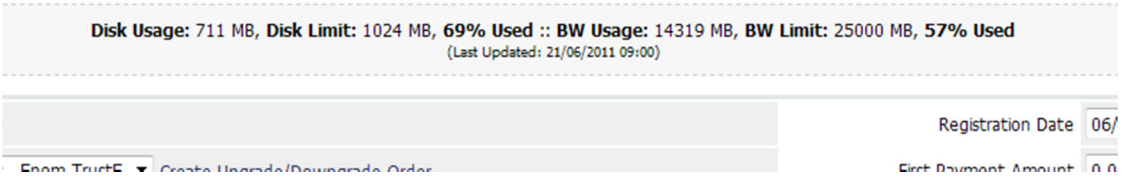
```
186 function template_extrapage($params) {
187     $pagearray = array(
188      'templatefile' => 'example',
189      'breadcrumb' => ' > <a href="#">Example Page</a>',
190      'vars' => array(
191         'var1' => 'demo1',
192         'var2' => 'demo2',
193      ),
194     );
195     return $pagearray;
196 }
197
```

A custom page defined in this way is linked to in exact the same way as calling a custom function. So within the client area output you would need to include a button or link with the modop specified as “custom” and the a variable set to the name you give that function, in this case “extrapage”.

The UsageUpdate Function

The UsageUpdate function can be used to perform a daily import of the disk and bandwidth usage for accounts from a server. The data imported is then used to display the usage stats both within the client and admin areas of WHMCS, and also in disk and bandwidth overage billing calculations if enabled for a product.

Admin Area Example



The UsageUpdate function is called by the daily cron if present in a module, for any active and enabled server, on a per server basis (important: it is called per server not per product). Therefore this can only be used if your module has a server created in WHMCS for it, products alone do not invoke it.

The function is passed the id, ip, hostname, username/hash, & password variables and is then expected to query the disk and bandwidth stats for all accounts on that server, and update them within the database directly all in a single call for speed and efficiency. An example of how to do this is included below.

```
201
202 function template_UsageUpdate($params) {
203
204     $serverid = $params['serverid'];
205     $serverhostname = $params['serverhostname'];
206     $serverip = $params['serverip'];
207     $serverusername = $params['serverusername'];
208     $serverpassword = $params['serverpassword'];
209     $serveraccesshash = $params['serveraccesshash'];
210     $serversecure = $params['serversecure'];
211
212     # Run connection to retrieve usage for all domains/accounts on $serverid
213
214     # Now loop through results and update DB
215
216     foreach ($results AS $domain=>$values) {
217         update_query("tblhosting",array(
218           "diskused"=>$values['diskusage'],
219           "dislimit"=>$values['disklimit'],
220           "bwused"=>$values['bwusage'],
221           "bwlimit"=>$values['bwlimit'],
222           "lastupdate"=>"now()",
223         ),array("server"=>$serverid,"domain"=>$values['domain']));
224     }
225
226 }
227
```

Admin Services Tab

Available in WHMCS V4.3 and later, the Admin Services Tab functions of a module allow you to define additional fields that should appear on the product details page of the admin area for staff use. These can be used just for informational output, or for settings and values stored in custom tables or remotely outside of WHMCS.

An example of what we use it for in the core WHMCS system is for our licensing addon module, where the license specific fields of the allowed domains, IPs and Directory can then be viewed and set from the product details view.

There is both an AdminServicesTabFields and AdminServicesTabFieldsSave function supported in the modules, with the former allowing you to define the additional fields to output, and the latter allowing you to handle any input in them on submission/save.

So on to an example, below we show you how you could define 4 extra fields that would show an input field, dropdown, textarea and info only output as examples, and then update them in a custom table of the database via the save event.

```
function template_AdminServicesTabFields($params) {

    $result = select_query("mod_customtable","",array("serviceid"=>$params['serviceid']));
    $data = mysql_fetch_array($result);
    $var1 = $data['var1'];
    $var2 = $data['var2'];
    $var3 = $data['var3'];
    $var4 = $data['var4'];

    $fieldsarray = array(
     'Field 1' => '<input type="text" name="modulefields[0]" size="30" value="'.$var1.'" />',
     'Field 2' => '<select name="modulefields[1]"><option>Val1</option></select>',
     'Field 3' => '<textarea name="modulefields[2]" rows="2" cols="80">'.$var3.'</textarea>',
     'Field 4' => $var4, # Info Output Only
    );
    return $fieldsarray;

}

function template_AdminServicesTabFieldsSave($params) {
    update_query("mod_customtable",array(
        "var1"=>$_POST['modulefields'][0],
        "var2"=>$_POST['modulefields'][1],
        "var3"=>$_POST['modulefields'][2],
    ),array("serviceid"=>$params['serviceid']));
}
```

Database Queries

You will have seen in various examples throughout these docs the use of database queries within module functions.

In all module functions, there will always already be an active connection to the WHMCS database. And you can run custom queries to obtain any additional data you need, or update/alter and data you need to using that connection without needing to connect through any custom code.

That can be done using the regular PHP/MySQL mysql_query() function or using the helper functions we have in WHMCS of select_query, update_query, insert_query which we'll provide a brief overview of below.

Select Queries

A select query is run as follows, with anything after the $where variable being optional. The $table should be the table name, $fields a comma separated list of fields to select, the $where var should be an array of criteria, $sort can be a field name to order by, $sortorder either ASC or DESC, $limits a range to select eg "0,1" "10,20" etc… and finally $join can be used for performing an inner join with another table.

So for an example:

```
$table = "tblclients";
$fields = "companyname,address1,city";
$where = array("id"=>$userid);
$result = select_query($table,$fields,$where,$sort,$sortorder,$limits,$join);
$data = mysql_fetch_array($result);
$val1 = $data[0];
$val2 = $data[1];
```

Update Queries

An update query can be run as follows, accepting a table name, array of fields to update, and a where clause for the criteria.

```
$table = “tblclients”;
$array = array(“companyname”=>”Test”);
$where = array(“id”=>$userid);
update_query($table,$array,$where);
```

Insert Queries

Insert queries are the simplest of them all requiring just the table name and an array of fields to insert. But it will return for you the ID of the newly created entry. So for example:

```
$table = “tblclients”;
$array = array(“firstname”=>”x”,”lastname”=>”y”,”companyname”=>”z”);
$newid = insert_query($table,$array);
```

Additional Resources

Here are some additional resources from our Developer Resources @ http://docs.whmcs.com/Developer_Resources that you might find useful when developing your module.

- **Internal API**
  This allows you to call various WHMCS functions and actions from within your module code and hooks
  http://docs.whmcs.com/API:Internal_API

- **Action Hooks**
  Action hooks allow you to hook into various events inside WHMCS and perform your own custom code as they occur
  http://docs.whmcs.com/Hooks

- **Addon Modules**
  Addon modules allow you to create modules that contain both admin area pages & output, as well as hook files
  http://docs.whmcs.com/Addon_Modules

- **Creating Pages**
  If you want to create custom pages for the client area then this shows you how
  http://docs.whmcs.com/Creating_Pages

- **External API**
  The allows you to call various WHMCS functions and actions from custom files and code outside of WHMCS
  http://docs.whmcs.com/API