Module developers guide for ZPanelX

Author: Bobby Allen (ballen@zpanelcp.com)
Version: 1.1

About ZPanel

ZPanel is an open-source web hosting control panel for Microsoft Windows and POSIX based operating systems. ZPanel is developed and maintained by the ZPanel Team. ZPanel is written in PHP and as standard works with the Apache Web Server and uses MySQL as for its backend data storage.

More information about the project and to download and get started with ZPanel, please visit the official website at: http://www.zpanelcp.com/

About the framework

The framework is considered the 'core' part of ZPanel, the framework contains multiple utility classes and handles the MVC architecture, the framework is what all modules sit on top of and communicate with. The framework was designed and developed by Bobby Allen in memory of his best childhood friend (David Dryden) who passed away in February 2011 at the age of 22, officially named the 'Dryden' framework it is designed to be super easy to develop modules with far greater integration for modules of that seen in previous version of ZPanel. The framework implements its own implementation of an MVC architecture which we feel provides greater security, easier integration and faster for developers to build robust and secure modules.

The purpose of this guide

The purpose of this guide is to provide an aspiring module developers with clear information on what the' Dryden' framework is, some of its functionality and how to begin writing modules for ZPanelX, we will also aim to keep this document as simple to follow as possible and is intended as a quick start guide rather than a full blown document.

Contents

Type of modules

In ZPanelX there are now three types of modules these are as follows:-

- System module
- User module
- Admin module

The main difference between these modules is their visibility, security and configurability. I will now explain how each of these module types differs.

System module

A system module does not include an interface; these modules generally only make use of system hooks and/or web services. System modules do not appear as an icon in ZPanel nor do they appear in the Module Admin module as these are generally very low level and do not require the kind of security configurations of other kinds of modules.

Examples of a system module may include any one of the following:- Nightly backup to remote a server, Send usage statistics to a web service every daemon run etc. both of which would make use of the hooks system.

User module

This is by far the most common type of module; a user module provides both access restrictions based on groups as well as the ability to enable or disable the module globally, user modules can also be moved between module categories.

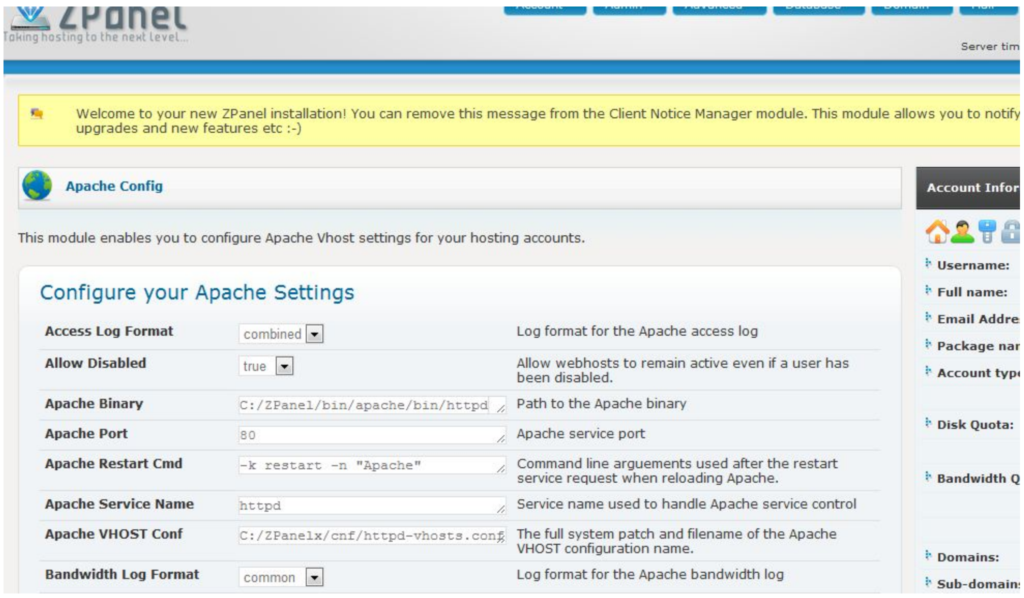
User modules provide full functionality thus meaning a web interface and can also provide web services and hook scripts.

Examples of this kind of module may include: Postgres Databases, SVN Repositories, Error Page Editor

Admin module

This module is designed to be used when user modules have configuration options and therefore Admin modules are only accessible from the 'Module Admin' module these should be thought of as configuration modules really.

An example of this kind of module is the *Apache Config* module show here:-



Welcome to your new ZPanel installation! You can remove this message from the Client Notice Manager module. This module allows you to notify upgrades and new features etc :-)

Apache Config

This module enables you to configure Apache Vhost settings for your hosting accounts.

Configure your Apache Settings

Access Log Format	combined	Log format for the Apache access log
Allow Disabled	true	Allow webhosts to remain active even if a user has been disabled.
Apache Binary	C:/ZPanel/bin/apache/bin/httpd	Path to the Apache binary
Apache Port	80	Apache service port
Apache Restart Cmd	-k restart -n "Apache"	Command line arguements used after the restart service request when reloading Apache.
Apache Service Name	httpd	Service name used to handle Apache service control
Apache VHOST Conf	C:/ZPanelx/cnf/httpd-vhosts.conf	The full system patch and filename of the Apache VHOST configuration name.
Bandwidth Log Format	common	Log format for the Apache bandwidth log

The standard folder structure of a module

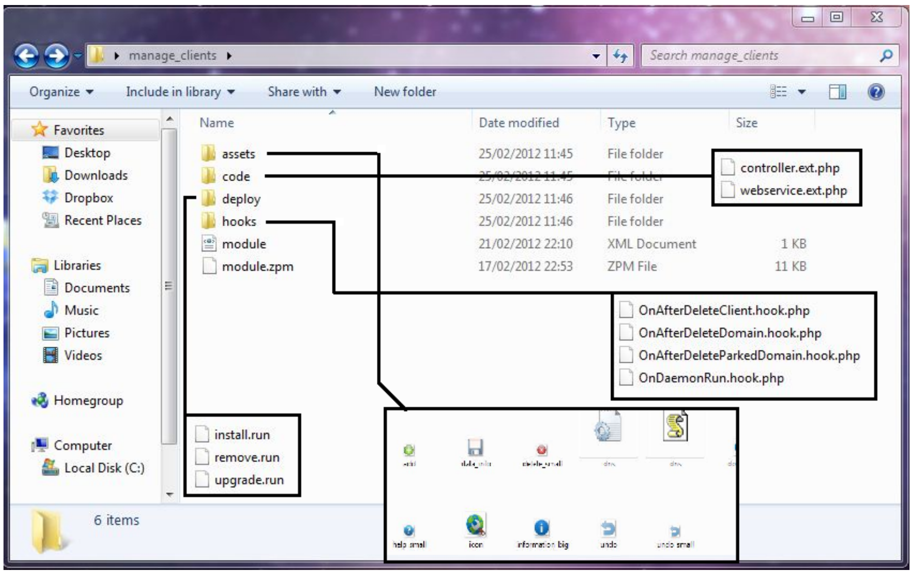
The example screenshot below shows an example layout of the main folders and files of a module.

The following files/folders are NOT required and only need to exist if you wish your module to make use of the more advanced features that the Dryden framework has to offer.

- code/webservice.ext.php
- hooks/*

For different types of modules you don't need certain files, for example a '*System*' module generally only uses hook files or has a web service extension therefore and as the '*system*' modules as explained in the '<u>Types of modules</u>' section don't have a graphical front-end therefore doesn't require the module.zpm file.

All modules MUST have a module.xml file.



Name	Date modified	Type	Size
assets	25/02/2012 11:45	File folder	
code	25/02/2012 11:45	File folder	
deploy	25/02/2012 11:46	File folder	
hooks	25/02/2012 11:46	File folder	
module	21/02/2012 22:10	XML Document	1 KB
module.zpm	17/02/2012 22:53	ZPM File	11 KB

I will now explain what the man folder types are used for and go into more detail about them.

The *assets* folder

The assets folder provides a standard place to store module specific images, CSS and JavaScript files. This folder is also a great place to store your modules icon that appears in ZPanel.

The files stored in this folder can be easily retrieved for use in the modules 'view' by using this template tag:-

```
<# ui_tpl_assetfolderpath #>
```

So we can easily display a module image that exists in the 'assets' folder like so:-

```
<img src="<# ui_tpl_assetfolderpath #>/myimage.jpg">
```

This will then display the image named 'myimage.jpg' on your modules user interface.

More information regarding use of template tags can be found in the Framework tinplating tags section of this document.

The *code* folder

The code folder contain executable PHP and is the first place that the Dryden framework will look for your module's *controller.ext.php* file.

The *code* folder also is home to the *webservice.ext.php* when in use. As well as these two standard files, the code folder is also the recommended folder to place in any PHP code libraries that you may want to use in your module and any other PHP includes etc.

More about the *controller.ext.php* file can be found later on in this document.

The *deploy* folder

If exists (this is not a required folder), The *deploy* folder is used by the zppy (pronounced 'zippy') client, the code folder can (if the module makes use of them) hold the following files:-

- install.run
- upgrade.run
- remove.run

These files are executed when the zppy client installs upgrades or removes the module from your ZPanel installation, although the technical aspects of how the zppy cient works are covered in the zppy documentation, these files are simply PHP files containing extra tasks (such as copying files, executing some SQL code, creating some extra folders etc.) to complete upon an installation, upgrade or removal of a module via the zppy interface.

The *hooks* folder

Again, this folder is not required and only exists if your module should make use of any hooks. The files (PHP classes) that exist in this folder are named with the Hook execution name and then with the *.hook.php* file extension.

More information on hooks can be found later on in this document in the section named 'Integration using hooks'.

The *module.xml* file

This file contains meta information about the module and in used in various places in ZPanel, this file contains information about the developer and the module itself, it includes an update service URL which is used by the server running ZPanel on to check and display if the module has updates available and if so will also provide a link to enable the user to download the latest version of your module.)

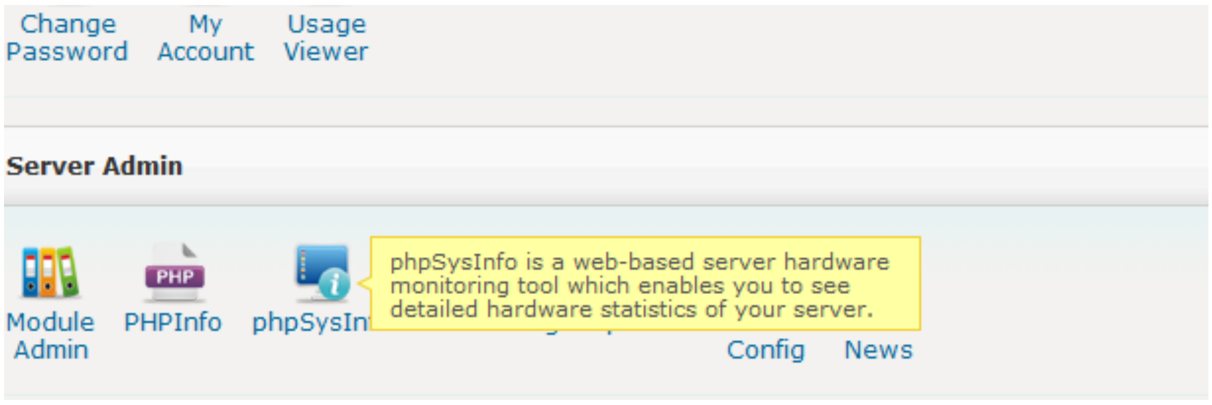
This file also contains the version number of the module as well as other information.

The main places where the contents of this file is used are as follows:-

- Import the module meta data into the *x_modules* table and uses the module description tag data on user however over when highlighting over a module. (See screenshot 1)
- The 'Module admin' module interrogates this file and displays the information when an admin clicks on the module information link. (See screenshot 2)
- The framework will also check for updates for the module using the content of the *<moduleupdateurl>* tag. (See screenshot 3)

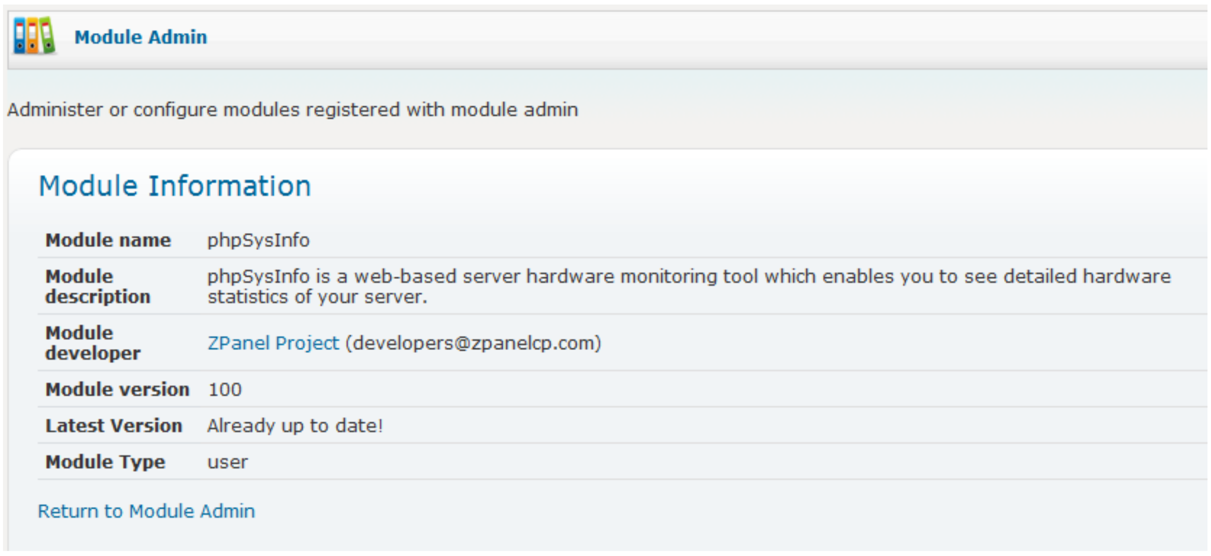
This file is also used in other parts of the system and obviously in future could also be used by third-party modules, this is most certainly a major file and one that as a module developer you should ensure that you configure correctly before releasing your module for public consumption.

Screenshot 1 – Shows the module description on 'hover over' of the module icon.



phpSysInfo is a web-based server hardware monitoring tool which enables you to see detailed hardware statistics of your server.

Screenshot 2 – Showing the 'module information' window which is part of the Module Admin module:



Module Admin

Administer or configure modules registered with module admin

Module Information

Module name	phpSysInfo
Module description	phpSysInfo is a web-based server hardware monitoring tool which enables you to see detailed hardware statistics of your server.
Module developer	ZPanel Project (developers@zpanelcp.com)
Module version	100
Latest Version	Already up to date!
Module Type	user

Return to Module Admin

The *module.zpm* file

In the MVC methodology, this file is considered the view. This file contains standard HTML/XHTML and can access data objects and code via. the template tags that are provided by the Dryden framework, for more information about the template tags, please see the Framework Template Tags section.

The *controller.ext.php* file

In the MVC methodology, this file is considered the model (controller extension). This file contains a PHP class with methods the methods from this file are accessed via the module.zpm file via the framework template tags, more information about these tags can be found in the Framework Template Tags section.

This file can also call and make use of any of the standard framework classes and methods under the *dryden/* directory, the Dryden framework will auto load any of these classes upon request. This functionality is provided by the controller.

Use the framework classes, do more with less!

The Dryden framework contains a whole host of easy to access static functions, think of these as utilities or 'tools' and where ever possible you as a module developer should make use of these in your modules instead of duplicating functionality.

By using the framework classes and methods, these are maintained by the ZPanel team and will mean you have less that you need to maintain etc as the ZPanel team will keep these core classes patched and if the low level PHP functions used are ever deprecated we'll replaced them to ensure that the functions remain working as they should.

Using an IDE (Integrated development environment such as NetBeans or Eclipse will allow you to easily browse and auto-complete class methods within your module code.

Great lengths have been taken to create utility classes for just about all general development requirements as well as class and method documentation conforms to the PHPDoc standard. A full list of the framework classes and their properties can be found online here:- http://developer.zpanelcp.com/api/

Don't break the user interface style; use the official CSS class names!

Although you don't have too, we recommend that you follow the official template CSS styles to ensure that your modules interface uses the standard CSS styles so that your module remains 'fluid' on new template/styles the user installs on their server.

This is covered in greater depth in the template documentation: - http://developer.zpanelcp.com/template/

The framework template tags

As a security feature execution of PHP has been disabled from being ran from inside the module.zpm file, all PHP code has to be placed inside a PHP method inside the module's *controller.ext.php* file, the use of template tags allows the model (the module's *controller.ext.php* file) to pass data back and forward to the view (the module's *module.zpm* file).

For module development there are four main template tag types which you can use, these are as follows:-

Translation tags:- `<:  Text to translate :>`

Core Template Tags: `<# ui_tpl_assetfolderpath #>`

Code execution:- `<@ MethodName @>`

Logic tags:- `<% if MethodName %>`, `<% else %>`, `<% endif %>`

Looping/Controls:- `<% loop MethodName %>`, `<~ ArrayKey ~>`, `<% endloop %>`

Each of the above tags with the exception of the translation tag relies on the *controller.ext.php* file of the current module.

It is important to note that each of the above tags MUST have a space after and before the tags are closed, not doing this will invalidate the tag and it will not be parsed, the tag will simply appear in the template without being parsed.

Translation tags

Translation tags are used for on the fly translation of text, the template translation tags does a database lookup in the *x_translations* table in the ZPanel database and if there is a corresponding language translation it will replace the text with the equivalent otherwise it will fall back and use the text string encapsulated between the '<:' and ':>' tag.

For example, this block of code in the module.zpm file would be successfully parsed and the translated string outputted to screen:

```
<: This is the text to translate :>
```

If the module developer provided translations for his module in French and the user had 'French' selected as their language via the 'My Account' module then the tag would be parsed and outputted in the template when parsed as:

```
Ceci est le texte à traduire
```

For those of you that cannot speak French, the above string simply says ' This is the text to translate'.

Be aware that for custom strings and less generic words and phrases a module developer will need to include new rows to the *x_translations* table to ensure his or her module is multi-lingual; this isn't required but a good idea if you want your module to have full localisation support.

Code execution tags

Probably the most used type of tag, this enables you to execute some PHP code and return some text output if required.

The requirements of this tag requires that you have a method (static function) in your *controller.ext.php* file, the method name is that of the tag contents prefixed with 'get', here is an example:-

```
static function getSayHello(){
      return 'I was told to say hello, so hello!';
}
```

To execute the code and get your text to display in the 'view', you would enter this tag wherever you wanted the text to appear in your *module.zpm* file:-

```
<@ SayHello @>
```

The above example is obviously really simple and your actual methods would normally return more than just a basic string and more likely to return an array or data object.

Logic tags

We also provide logic tags so a decision can be made at the view layer as to display a module related area or text, these logic tags basically just check the return value of a static function in your modules *controller.ext.php* file, here is an example:-

So a simple static function in our *controller.ext.php* file:-

```
static function getTodayIsMonday(){
      if(date('D') == 'Mon')
            return true;
      return false;

}
```

The above code, simply checks if the day today is Monday, so we can now use this logic in our module.zpm file like so:-

```
<% if TodayIsMonday %>
      Yay today is <strong>Monday</strong>
<% else %>
      I can assure you that today is not Monday!
<% endif %>
```

You can also just use a logic block without the use of an *<% else %>* tag, so for example:-

```
<% if TodayIsMonday %>
      <@ ShowMeMyJobs @>
<% endif %>
```

The above code would only run the *<@ ShowMeMyJobs @>* code if the day is a Monday!

Control loop tags

A great use for control loops can be when you want to generate a drop down menu from a database table or if you want a list of rows to display. With the control loop tags you can easily iterate over an array of results.

Like the Logic and Execute code tags, the method needs to be prefixed with 'get'.

Control loops require the method to return a valid data array, if a valid data array is returned you can then use the results in the template like so:-

```
<td>Domain name</td>
<td>Domain home directory</td>
<% loop AllDomains %>
<tr>
   <td><~ domainname ~></td>
   <td><~ domaindir ~></td>
</tr>
<% endloop %>
```

The result would then create a row and display all of the domains that exist in the returned array.

Posting of HTML form data

When posting data using forms inside your modules you need to use an 'action' URL parameter, the value of which then passes the data to a method in your modules *controller.ext.php* file for you to then manipulate, the method name should be prefixed with 'do'.

Here is an example snippet HTML form tag:-

<form action="./?module=cron&**action=DeleteCron**" method="post">

I've highlighted the form action to class method part, when the form is posted, the data is sent back to the main controller and then on to the module's *controller.ext.php* file this is where it will execute the contents of the method named 'do**DeleteCron**', this should now make sense and you can see where we have used the 'do' prefix for posting data in the same way that we use the 'get' prefix for when we request data from the view layer using the '<@' tags. The result of which should be an easy to browse *controller.ext.php* file where you (as a developer) can quickly differentiate between the method types.

Grabbing field data

Inside your 'do' method (for example '*doDeleteCron*' as shown in our above example) you can easily grab the field data from the inputs you 'posted' from the forum using this controller method:-

```
global $controller;
...
$controller->GetControllerRequest('FORM', 'FormFieldName');
```

Using the above example you'd replace the *FormFieldName* text with the input field's name to quickly grab the posted form data.

Another example of getting the raw form data and processing it inside you 'do' method could look like so:-

```
if ($controller->GetControllerRequest('FORM', 'inScript') == '') {
   self::$blank = TRUE;
   $retval = TRUE;
}
```

The above example would get the value of the text field name '*inScript*' from the posted form and then check it to see if its blank, if it is blank it sets a static variable '$blank' to 'true' which can then be checked before returning.

The module update service

By default ZPanel will go through on a daily basis and check the versions of all modules that exist in the */modules/* folder in ZPanel, if the module's *module.xml* file is found to have a valid *<moduleupdateurl>* tag, ZPanel will connect to the remotely *hosted updateservice.xml* file at the given URL.

If the module URL is not contactable then the module will simply report as being up to date (there are no updates available) on the assumption that the module developer no longer owns the domain name or no longer develops the module and therefore we assume that the user is using the latest version of the module.

The *updateservice.xml* file can be manually updated by hand and be a static XML file or depending on your skills as a developer you could automate this with PHP or another dynamic language if you wanted too.

This is what the contents of the updateservice.xml file looks like:-

```
<?xml version="1.0" encoding="utf-8"?>
<updateserver>
<latestversion>100</latestversion>
<downloadurl>http://packages.zpanelcp.com/example_module.zpp</downloadurl>
</updateserver>
```

The *<latestversion>* tag should contain the latest version of the module that you have released for public use, this will be used by ZPanel to display and determine if the currently installed version of the module is older than your most recent release, if the version number in the *<latestversion>* tag is newer to that of the one that the ZPanel server has installed it will then use the link in the *<downloadurl>* tag to provide the ZPanel administrator with a download link to grab the latest version.

Integration using hook files

Hooks are a new feature added in ZPanelX which enables modules to 'hook in' at a given point, the framework and core classes provide several default hook points of which can all be used in any type of module being that a *User*, *System* or *Modadmin* module.

Hook files that you create must reside in your */hooks/* directory of your modules root folder and must be named with the hook reference and then have the *.hook.php* file extension.

Hook files are standard PHP scripts that are executed inline, from a technical view; these files are simply 'included' at the point of execution.

So for example, if you wanted to run some code when a user logs out of ZPanel, you would need to create a file named *OnUserLogout.hook.php* the contents of this file would then be executed each time a user logs out of ZPanel, a practical use for this hook might include to destroy other server sessions for other web apps such as phpMyAdmin etc.

There are ton's of standard hooks across the ZPanel framework and as we are always adding additional hooks we don't supply a complete list of all the available hooks (there are so many!) and instead we recommend that developers explore the core code and see where they need a hook to execute and then choose a hook name suitable to their requirements to use from the code.

Adding hook points into your modules

In more advanced modules you may wish to provide your own hook points to enable other developers to integrate easier with your module, to do this you simply add this declaration (in the same way we have in the framework) to your *controller.ext.php* in the desired method at the point of required execution.

```
runtime_hook::Execute('YourHookNameHere');
```

When adding your own hook points, be sure to give them a sensible but unique name as someone else may accidently use the same name as your hook and in which case may have undesired affects.

Adding XMWS web service functionality to your module

If you have designed your module to use 'interface' methods and 'worker' methods as described in The Best practices of module development for ZPanelX then adding XMWS functionality really is quick and easy.

The XMWS web service layer is documented in this document:
http://developer.zpanelcp.com/xmws/

Debugging your modules at runtime

To view errors within your modules you are advised to enable PHP error reporting if you haven't already done so in your *PHP.ini* file therefore when your module throws an error you'll receive an error message on screen; Production servers do not normally have PHP error reporting enabled for security reasons therefore unless you enable this you are more than likely to receive a blank screen or for it to simply suppress the error.

By appending the current URL of a page with &**debug=true** you will then notice that debug information containing how much memory the current page is using, how long it took to execute as well as a list of all modules that are currently loaded can be seen at the bottom of the current page.

You can also use the global $zlo (ZPanel Logging Object) within your modules *controller.ext.php* file to log errors to the predefined data logging medium as set in the *x_settings* table. (eg. *Logfile*, *database*, to *screen* or *email*) like to:-

```
global $zlo;
...
$zlo->code(3445); // A custom error code, whatever you want!
$zlo->detail('My module error description would go here');
$zlo->mextra('This can be used for extra logging information such as a trace log
etc.');
$zlo->writeLog();
```

Best practices of module development for ZPanelX

When designing your module you should think about what methods you need and what ones do what, ideally you should split your methods up into 'worker' methods and 'interface' methods by doing this you can easily make use of your 'worker 'methods not only from the module web interface but also using XMWS (Web services) this will reduce the amount of code and generally keep things clean and tidy.

A 'worker' class basically does the actual action and takes parameters and relies on the' interface' class, hook script or *webservice.ext.php* to do any error checking or validation.

Always make sure that your module’s *module.xml* file is configured correctly and up to date.

Its best to keep HTML and PHP code separate therefore we do not recommend you echo out HTML from your PHP methods but understand if you want to and in some circumstances we are aware that it is necessary to do so! – The new template logic tags and conditional loops in most cases will enable you to have a dynamic bridge between your code and module.zpm (HTML/XHTML content).

When releasing your modules to the public ensure that you have fully tested your module, its always good to provide documentation too, it just makes things more ‘professional’.

If you want your modules to have a maximum audience and to have full compatibility with the *zppy package manager* you must ensure you package your module into a *.zpp* (ZPanel Package File) file, more information about how to achieve this an thus provide automated updates and a means of installing your modules automatically from both the *Mod Admin* module and the CLI read the *Zppy documentation* which can be found here:- http://developer.zpanelcp.com/zppy/

The best way to learn

By far the best way to learn and one of the beauties of open-source software is to look and explore the code, for a module developer wanting to write a module for ZPanel you should take a look at the default modules that come with ZPanelX, get a feel for what they are doing - the ability to look at live working examples and armed with this guide you should now be well on your way to writing your first module.

Other resources

The official PHP.net website provides outstanding documentation for the language if you are a professional developer then you don't need us to tell you that the PHP.net website is a great resource and tool to help you build any type of module.

The official ZPanel community forums are also a great place to ask questions and get help from the official ZPanel development team

Developer.zpanelcp.com provides full class and method documentation in an easy to browse format, this is great to a quick look at all available framework classes and methods – especially if your IDE (Integrated Development Environment) of choice doesn't support PHPDoc auto-completion.

All good developers should use a decent IDE, Personally I recommend using Netbeans, its open-source, comes with version control support, auto-completion and full of other really useful tools.

If you intend on making your module professionally use version control, sign-up for an account on GitHub.com if you don't already have one and use your Github repository to document, host your source code and provide you with bug tracking functionality! - Github.com is the way to go!