# Zen Cart Developer Documentation

**Zen Cart Team**

**Jul 24, 2019**

# Contents

**Warning:** The documentation here is very much work in progress. It will change as code for v157 evolves.

This site contains documentation related to the Zen Cart e-commerce system, and other projects related to Zen Cart.

The site is organized into a number of Manuals.

The currently available Manuals are:

- *Contributing To Zen Cart*
- *Developer Documentation*
- *Habitat Developer Environment*
- *Docker Development Environment*

You may contribute to this documentation via our Github Repository.

CHAPTER 1

# Contributing To Zen Cart

## 1.1 Getting Started

### 1.1.1 But What Can I Offer?

Your experiences will be both the same as hundreds of others and yet uniquely yours at the same time. Your own unique perspective, combined with your own creativity, coupled with your skills and experience, mean you have something to offer!

See a bug? Know of an improvement, even if it's tiny? Feel free to share about it!

### 1.1.2 I'm Scared! This is my first time!

That's completely understandable. We all started with a sense of excitement and fear all at once. Excitement to be a contributor, and fear that my contribution might be wrong or break something or be rejected. Those are normal feelings. Here's an excellent article explaining those same feelings and showing how easy it is to get through them!

### 1.1.3 Where do I start?

It's really as easy as 1-2-3 . . .

1. Start with reviewing the *Reporting Bugs* document, as a guide to first steps about exploring the issue further, checking whether it's already being addressed someplace, whether it's a core bug vs a "bugs-me bug", along with how to report the issue to the dev team and community for further discussion using Github Issues or the Zen Cart Support Forum.

2.If you're simply "looking for ways to get started", take a look at the existing Zen Cart Issues on Github for issues that are tagged with "up-for-grabs" or "first-timers-only".

3. If you're going to contribute code, read through the *Pull Requests* document to learn more about setting up git and github, cloning the repository, creating a branch in which to make your code changes, and how to submit pull requests. As you start working on your code changes and pull request, review the *Coding*

*Standards* guide to ensure that your code matches both the formatting and testing requirements. If you're submitting a new feature and know how to write PHP code tests, please be sure to submit tests with your code, where appropriate.

## 1.2 Coding Standards

### 1.2.1 Old Standard

Legacy Zen Cart code has used a modified *phpBB* coding style, with notable characteristics such as "indent with 2 spaces, not tabs", and "The opening brace (ie: *{*) for *class* and *function* definitions, as well as *if/else* statements, was kept on the same line", etc.

### 1.2.2 New Standard: PSR-2

In the interest of progressively modernizing the code, going forward we are adopting the PSR-2 coding standard (which also includes PSR-1 standards).

#### 1.2.2.1 Difference from the "Old Standard"

To be clear: PSR-2 uses "4 spaces instead of tabs", and puts the opening *{* braces on a new line when used with *class* and *function* declarations (but keeps them on the same line when used with controls structures such as *if*, *for*, *foreach*, *switch* or *while* loops.)

For example, the old style would be:

```
class foo extends baz {
  function bar() {
    if ($var1 == $var2) {
      // do something
    }
    return;
  }
}
```

and the new PSR-2 equivalent would be:

```
class foo extends baz
{
    function bar()
    {
        if ($var1 == $var2) {
            // do something
        }
        return;
    }
}
```

### 1.2.3 When to use PSR-2

So, new code should use the PSR-2 standard. That is, when new files are created such as new functions or class files, they should use the PSR-2 standard.

### 1.2.4 When to use old style

When doing maintenance to existing code in existing files, and to some degree when adding code to existing files, the format of those files should not be radically changed.

Reasons: difficulty for people upgrading since they need familiar comparison points to merge changes, and familiarity for those working in those files ... it's better for there to be one style per file.

### 1.2.5 When to reformat an entire file

If a file needs reformatting, that reformatting should be a SEPARATE commit (and preferably separate Pull Request) from ANY other changes. ie: there should be no functionality changes, no changes to the file other than white-space.

This is for the purpose of maintainability, and controlling understanding of the flow of code changes ... especially important if a bug is later found in the code.

### 1.2.6 Comments in Code

It is appropriate to use comments to explain what is happening in a given section of the code. This is so that other programmers coming after you (and you included!) can quickly understand intended the logic.

The standard phpDocumentor coding standard (ie: common PHP DocBlock syntax) is the preferred approach, as well as inline comments approximately every 10 lines of code.

### 1.2.7 What about files containing both HTML and PHP?

Where possible, (new) code should always output HTML/CSS/JS directly, and NOT use PHP to echo the HTML.

Code-editing programs can properly parse and display HTML/CSS/JS if they are entered in "raw text" - but they can't if they are "echoed" via PHP.

Examples:

#### 1.2.7.1 Good:

```
` <div class="xxxxxx"><?php echo sprintf(LANGUAGE_DEFINE, ...); ?></div> `
```

#### 1.2.7.2 Bad:

```
` <?php echo ' <div class=xxxxxx">' . sprintf(LANGUAGE_DEFINE, ...) . '</
div>'; ?> `
```

### 1.2.8 Separating content, markup and logic

Where possible, try to keep these things separate:

- Files in *includes/languages* should contain strings (not HTML/CSS markup);
- Files in *includes/modules* and *includes/functions* should be where logic resides (not markup);
- Files in *includes/templates* should be where display markup resides (not extensive logic).

### 1.2.9 Namespaced Autoloading

The PSR-4 autoloading standard is used for handling code in the ZenCart namespace (ie: the files in */includes/library*).

## 1.3 Pull Requests

This article is intended to help provide a basic understanding of contributing to forked repositories on github.

### 1.3.1 Quick Reference for Newcomers to git or github

For the basics of learning git, see: https://help.github.com/categories/bootcamp/

For the basics of learning github, see: https://help.github.com/articles/good-resources-for-learning-git-and-github/

To collaborate by sharing your code suggestions, you will need to "fork" our public repository. For the basics of understanding forking and how to manage that fork, see: https://help.github.com/articles/fork-a-repo

If you're looking for paid training courses on using git/github, see: https://training.github.com/

Glossary of common terms in git and github: https://help.github.com/articles/github-glossary

### 1.3.2 Client Applications to Interact with git

To interact with git, you can use the command line or you can use a desktop application. (There are also mobile/tablet apps, but we won't cover that here.)

We recommend working with one of the following first two options.

- command line. Purists will use the command line, for reasons they will pine eloquently about if asked. There are hundreds of resources online showing how to use all the git features via command line, starting with the links at the top of this page.
- **SourceTree for Windows and Mac - this is another good free desktop application** which does a good job of giving you access to all the power of git/github/bitbucket/mercurial in an easy visual interface. We highly recommend using SourceTree due to its simplicity.
- When you first set up SourceTree, it will ask you to log in to your Atlassian account. The account is free, and no side-effects, and no personal information needs to be provided besides name/email.
- There's another option, which we're NOT recommending here, as its UI is very complicated. Also (at the time of this writing) it has an important limitation when working with forked repositories: It doesn't support multiple remotes, which means you'll need to use the command line to keep your checkouts up to date with our central repository's branches else you'll have problems when issuing pull requests. That other option is github's own application: Github Desktop for [Windows](https://windows.github.com/) or [Mac](https://mac.github.com/) - these are free desktop applications written by Github, and make visual interaction **easy when working only with your own repositories but not when regularly contributing to someone else's projects**.
- While you won't find us documenting how to use them, there are several other git GUI client apps

### 1.3.3 Workflow

Here's the workflow you'll want to use as a collaborator with Zen Cart® and probably most other open source projects:

#### 1.3.3.1 Basic setup

1. github account Create a github account at https://www.github.com
2. Fork the Repository On the github site, fork the project's repository (at the time of this writing, the main ZC repository is at https://github.com/zencart/zencart). This makes a copy of that repository into your own github account, which is where you will upload your changes since you only have "write" permissions in your own repository.
3. Clone the repository to your own computer. (see below)

#### 1.3.3.2 Cloning a repository

- Get the URL for the repository you're cloning

1. Go to **your** github account page
2. Go to **your** fork of the Zen Cart repository: *https://github.com/YOURNAMEHERE/zencart*
3. Halfway down the right side of the page you'll see a URL in a textbox, with a clipboard icon next to it. It probably says "HTTPS clone URL" above it. You'll want to click the clipboard to copy that URL to your computer's clipboard.

Now switch to your desktop application of choice and use that URL to clone the repository locally:

- command line:

1. pick a working directory on your PC
2. cd into that directory
3. type: *git clone PASTE-THAT-URL-HERE* . (note the . at the end, which says "put it in the current directory")

- SourceTree:

1. Choose File, New Clone...
2. In the Source Path/URL, paste the URL you copied from github
3. For the destination path, pick a folder on your PC
4. For the bookmark name, call it whatever friendly name you want to remember this repo by. It will show up in [SourceTree](http://www.sourcetreeapp.com/)'s bookmarks list of repositories you've got.
5. Click Clone or OK to have it start the clone. It will take a few seconds for it to download the repository contents to your computer.

#### 1.3.3.3 Add an upstream remote

To keep your own fork up-to-date, you'll need to periodically merge updates from the main Zen Cart repository. This involves telling your own local (on your PC) git repository about the main Zen Cart repository location. To do this you must add what's called an "upstream remote". (See the glossary link above for what a "remote" is.)

- command line:

1. cd into the directory of the repository you're adding the remote to
2. type: *git remote add upstream https://github.com/zencart/zencart.git*
3. type: *git fetch upstream*

- [SourceTree](http://www.sourcetreeapp.com/):

1. Choose Repositories from the main menu
2. Choose Repository Settings...
3. Click Add
4. For Remote Name, use: **upstream**
5. For the URL use: *https://github.com/zencart/zencart.git*
6. For the github username, enter your own github account name
7. Click OK
8. Click Fetch

#### 1.3.3.4 Making a working-branch

Any time you're going to contribute code changes, you'll want to first make a working branch. (For more on branches, see the glossary and other git references at the top of this page.) Branches are how git keeps different versions of changes separate from each other until such time as someone approves merging them together into the main branch.

#### 1.3.3.5 Pick a branch name

Decide on a new branch name. (The branch name should be brief, but meaningful; ideally a max of 6 words, all hyphenated, no spaces.)

#### 1.3.3.6 Make the branch

Now use that name for name-of-your-branch-here, below:

- command line:

1. cd into the directory of the repository you're intending to make changes to
2. In this example we'll be branching from the develop branch
3. Type: *git branch name-of-your-new-branch-here develop*
4. Type: *git checkout name-of-your-new-branch-here*

- SourceTree:

1. First, make sure you're in "Log View" (View, Log View)
2. Find where it shows *upstream/develop*, and right-click on that row. Choose Branch... from the pop-up menu
3. Give it the new branch name
4. Leave the "checkout new branch" box checked
5. Click Create Branch or OK

#### 1.3.3.7 Make your code changes

- Edit or create whatever files are applicable to the changes you wish to submit for consideration.
- Test your code. Test to make sure your changes work, and that you've not broken anything else in the process.
- Once your code is ready for submission, you'll need to commit the changes, and push them to your github account and then create a Pull Request. Those steps are described below.

- MAKE SURE YOUR CODE COMPLIES WITH ZEN CART CODING STANDARDS, else it may be rejected.
- If your code can be tested with phpunit, be sure to include those tests in your commits and pull request.

#### 1.3.3.8 Commits

To commit your code, you must first "stage" the files which are to be included. See the git docs mentioned at the top of this page for more detailed explanation of what this means.

Once you've staged the files, then you commit them, which saves that group of changes together.

You can make multiple commits (that is, stage the files and commit them) towards any given issue. This allows you to make numerous smaller commits which are easily described in connection with the specific files that relate to those smaller changes.

- command line:

1. cd into the directory of the repository you're committing from
2. type *git status*
3. this will give you a list of changed/added/deleted files
4. type: *git add filename1.php filename2.php* (and any other files, etc)
5. type: *git commit*
6. This will pop up your text editor where you can supply a commit message. See explanation of commit messages in the sub section below.
7. Save the message using whatever method your text editor uses to save-and-exit
8. This will have the commit saved locally. You can continue working and making more commits until you're ready to push them all up to github (see pushing commits below)

- SourceTree:

1. First, go into File Status view. Click on "Working Copy" in the left nav menu under File Status. Or, use the View menu and choose File Status View.
2. Here you'll see a list of files on-screen which have changed in some way (edits, adds, deletes)
3. You can also see exactly what's changed by clicking on those files and viewing the "diff" on the other side of the screen.
4. For each file that you wish to include in the current commit, highlight it in the bottom part of the window, and click the Stage Selected button on the button-bar. It may ask you to confirm that you wish to Add it. (The Windows version has a tiny up-arrow that lets you do the staging as well, instead of using the Add from the top button-bar).
5. Once your files are all staged, click the Commit button in the button bar
6. This will open a dialog where you can supply a commit message. See the guidance around commit messages in the next section below.
7. Click the commit button in the bottom right.
8. Your commit is now saved locally on your PC. You can continue making more commits until you're ready to push them all to github, as described below.

#### 1.3.3.9 About Commit Messages

1. The "subject" or "first line" of a commit message should be no more than 50 characters.
2. The next lines can have as much detail as you like. Consider using [Github Markdown syntax](https://help.github.com/articles/github-flavored-markdown) for any formatting you might wish to include in the message. Feel free to use blank lines, and even use hyphens to create bulleted lists (hyphen plus a space)
3. If you're contributing code to help with an "Issue" that's already listed on the [Zen Cart github Issues](https://github.com/zencart/zencart/issues?state=open) page, include that issue number in your commit message, with the hashtag in front of it, like this: #101 for issue number 101.
4. Further to the point above, if your commit ["fixes" or "closes" or "resolves" an existing open issue then include the word "Fixes" before the issue number](https://help.github.com/articles/closing-issues-via-commit-messages/), ie: "Fixes #101" somewhere in your commit message. This will cause Github to close the "issue" ticket when your pull request is merged, and helps keep things tidy.
5. If you're committing code that addresses a bug reported on the Zen Cart support forum, include the URL for that bug from the forum, so we can cross-reference it.

Suggested reading: 7 Principles for Good Commit Messages

#### 1.3.3.10 Pushing Commits To Github

Now that you've made some commits to git on your local PC, you must push them to (your account on) github in order to prepare to share them.

- command line:

1. again, cd into the directory of your working repository
2. type *git push origin name-of-my-working-branch*

- SourceTree:

1. Click the Push button in the top button bar.
2. From the pulldown for "Push to repository", be sure that "origin" is selected. That's **your** github repository, and you must push to there.
3. Next make sure you check the box next to the branch you've been making your commits in. Uncheck all the others.
4. Click OK
5. That's it! Now all the commits you've made in that branch on your PC will show up in your Github account.

#### 1.3.3.11 Pull Request

(You'll also see *Pull Request* referred to as a *PR*)

After you've pushed your working branch (ie: containing your new commits) to your own Github account, you will need to create a [Pull Request](https://help.github.com/articles/using-pull-requests/) in order to ask the Zen Cart developer team to review it and consider it for inclusion in core code.

You'll do this from your browser:

1. Go to your Github account in your browser.
2. Go to *your* zencart repository.

3. You will see a green "Compare and Pull Request" button. Click it. (If it's been several hours since you did the push, it might not show the green bar. In that case, click the 'Branches' link, where you will see a Pull Request button next to each of your branches. Click the one next to the branch you want to do the pull request from.)
4. Now you can review the collection of commits and file changes, and add a descriptive message to the pull request. If you're fixing something that's already got an open issue for it, be sure that the issue number is included in your Pull Request message. ie: #101. If you believe your Pull Request fully fixes the open issue, then say "Fixes #101", as the keyword "Fixes" helps do proper cleanup of tickets once closed.
5. TIP: If you are contributing to Zen Cart develop, be sure that your pull request "compare" is indeed being compared against the *zencart:develop* branch.
6. Click the next green button to Create Pull Request
7. Now you wait for others to review your code. The Zen Cart developers (and anyone else who has clicked "Watch" on the Zen Cart main repository) will get an alert about the pull request. Anyone wishing to reply with their opinions of what you've submitted can engage in dialog with you and one another while the code is reviewed.
8. If your code hasn't complied with the coding standards, or has bugs or is incomplete, you may be asked to submit more commits to rectify the problems. In that case, you will repeat the steps above for making code changes, making commits, and pushing those commits to github. As long as you push to the same branch on github, then all those commits will automatically be included in the pull request, so reviewers can see the updates you push.
9. Once the core team decides what to do with it, they have basically three options: to accept it (merge the pull request) or defer it (until a later date) or reject it (not merge it and close the pull request). Github will automatically email you about all updates and comments made about your Pull Request. (NOTE: Your PR won't be merged until you have signed the CLA.)

#### 1.3.3.12 Keeping Your Local Branch Current

When you or others make pull requests that are accepted into the Zen Cart core repository, that will make your own local copy be outdated. To keep current, you must periodically bring in the changes from the "upstream remote" we created earlier. (see: https://help.github.com/articles/syncing-a-fork/)

- command line:

1. type: *git fetch upstream*
2. type: *git checkout develop* (ie: if you're going to pull changes from the *develop* branch)
3. type: *git merge upstream/deveop* (merges your local copy with the upstream remote)
4. type: *git push* (brings your forked repository up-to-date)

- 'SourceTree <http://www.sourcetreeapp.com/>'_:

1. Click the Pull button in the top button bar
2. For Pull From Repository, choose "upstream" from the pulldown menu
3. For Remote Branch To Pull, choose "develop"
4. Leave the "commit merged changes immediately" box checked, and the others unchecked.
5. Click OK

#### 1.3.3.13 Cleaning up old branches, or grabbing new branches

From time to time you and others will add or remove branches from the github repositories, and you will want to keep your PC in sync with those.

- command line:

1. type: *git fetch upstream*

- SourceTree:

1. Click the Fetch button on the button bar
2. There are 3 checkboxes. Check them all. (You could opt to not prune/delete any local branches you've created, if you want to preserve them to understand your own work history, by unchecking the corresponding box.) (You could also fetch from individual remotes manually, and prune only when fetching from upstream, but never prune when fetching from your own github master)
3. Click OK

### 1.3.4 References

There are many more great resources explaining how all of this works. Some which you might wish to review include:

- git protocol by thoughtbot
- code review process

*The SourceTree name is copyright Atlassian. Zen Cart receives no compensation or consideration for recommending SourceTree; we simply find it to be an extremely capable and useful app for beginners and novices alike.*

## 1.4 Reporting Bugs

We welcome bug reports. Please ensure your bug reports align with the following standards:

1. A clear description of the symptom, ie: the actual "problem"
2. Clear specific (preferably numbered) steps which can be taken to recreate the problem consistently.
3. If there are any pre-requisites needed to make the problem possible, such as configuring a certain Tax Rate for a specific State/Country, etc, those need to be clearly listed.
4. Clearly state what you believe it "should" do, and why. If relevant, provide a URL to a resource which confirms the standard ... for example a URL to the correct address-format of a certain country if you're reporting that the formatting is wrong in ZC for that country.
5. Provide screenshots if possible or relevant.
6. If the bug has been discussed on the Zen Cart support forum, include the URL to that discussion thread.

### 1.4.1 Posting the Issue on Github

To post your issue on github:

1. Login to Github
2. Navigate to the Zen Cart repository "Issues" page
3. Check whether your issue has already been reported. If so, click on that existing issue and add a comment to it, explaining exactly how your situation is similar.
4. If your issue isn't already reported, click on the "New Issue" button and supply the info explained above. Be thorough and as detailed as possible.

### 1.4.2 Issue Labels

After you have posted an issue on Github, a Team Member may assign various labels to your Issue.

The current labels are :-

---

**Note:** @todo update issues

---

1. Code Location
    1. Admin
    2. Catalog
    3. Checkout
2. Issue Type
    1. Possible Bug (Where you have suggested that a bug exists, but this has not been confirmed)
    2. Confirmed Bug (where someone else has provided confirmation of the bug)
    3. Enhancement
    4. Code Reformatting
    5. Feature Request
    6. Question
    7. Refactoring/Optimizing
    8. Styling (where it relates to output html and/or css)
3. Issue Status
    1. Up For Grabs (not assigned to a particular user)
    2. First Timers Only (for simple fixes to encourage those that have not committed before)
    3. In Progress ( Normally automatically added when a PR references an issue)
    4. Duplicate (Should reference the duplicate issue)
    5. On Hold
    6. Won't Fix

### 1.4.3 Reference Guide

More information on Github Issue Reporting

## 1.5 Documentation

This documentation is built using Sphinx and RestructuredText

RestructuredText is similar to Markdown, but adds some extra bells and whistles that allow for much easier building of structured documentation.

In order to properly contribute to the documentation project you will need to install some additional software.

### 1.5.1 Installing Documentation

The documentation can be installed from github.

i.e.

*git clone https://github.com/zcwilt/zc-dev-docs.git zen-docs*

This will clone the documentation into a directory called *zen-docs*, you can of course change the directory name the documentation is cloned into.

### 1.5.2 Installing Sphinx

Please see the Sphinx Documentation for installing the software on your computer.

### 1.5.3 Building Documentation

After editing any files you want to add/change , you will need to run Sphinx to rebuild the documentation.

You will first need to change into the directory where the documentation is stored, i.e. the directory used in the *git clone* command.

From the command line you should then run *make clean html*

This will build the documentation in the *_build/html* directory, and you can open the index.html file in this directory to preview the documentation.

### 1.5.4 Resources

## 1.6 Unit Testing

> **Warning:** The documentation here was originally written for v2, and needs updating for v157

Writing automated tests is important to ensure that the tested features function properly both before and after making future changes to the system.

### 1.6.1 Preparation / Initial Setup

To prepare to run the Unit Test test suite:

1. First, install Composer:
   - go to *getcomposer.org*
   - download and install
   - follow the Getting Started instructions.
2. Run *composer install*
3. Add *vendor/bin* to your path.

### 1.6.2 Running Tests

Run all tests, using:

*phpunit -c testFramework/unittests/phpunit.xml*

### 1.6.3 Running Individual Tests

While you are debugging a specific test (such as one you're adding, see below), you may want to just call it in isolation rather than running the entire test suite. The syntax for doing this is:

*phpunit –filter <test class name> <path to test class file>*

For example,

*phpunit –filter testPaginationCase testFramework/unittests/testsPaginator/paginatorTest.php*

### 1.6.4 Adding Tests

To add a new test, create your test in an appropriate subdir under *testFramework/unittests*.

### Test Tips * Note that if you have defined constants in your file, you may have to use the trick of setting the class global variables as is done in *testFramework/unittests/testsSundry/AdminLoggingTest.php*:

```
protected $preserveGlobalState = false;
protected $runTestInSeparateProcess = true;
```

- If you want to put your tests in a new folder at the level of *testFramework/unittests*, remember to add the new foldername to *phpunit.xml*.

### 1.6.5 Web Tests

Some features can only be properly tested by running them through a browser. We do this using Selenium and Firefox.

### 1.6.6 Preparation and Setup for Web Tests

1. Follow the Prep/Setup above for Unit Testing, as these are dependencies for the Web Tests
2. Install a Java JDK if one isn't already on your computer.
3. Download Selenium Server Standalone Java Agent. This will give you a *selenium-server-standalone-<version>.jar* file.
4. Install Firefox if not already present.

Be aware that sometimes the "latest" Firefox may not work with Selenium ... so occasionally may need to use an older Firefox version, or upgrade Selenium to a newer version.

### 1.6.7 Preparing custom configurations

WebTests need to run from a local webserver, so you need a MySQL+Apache/Nginx (or equivalent) setup already present on your computer, and need to be able to access it from your installed Firefox browser. The Zen Cart Habitat server is ideal for this.

Since the webtests will run zc_install, be prepared for your database to be wiped out and your configure.php files to be updated.

In the *testFramework/config/* folder there is a *localconfig_EXAMPLE.php* file. Make a copy of that file and configure it to suit your local environment:

1. filename: *localconfig_YOURUSERNAME.php* ... ie: if I login to my PC as *bill* then the filename would be: *localconfig_bill.php*
2. Inside the file, edit the *define* statements to reflect your local webserver's domain name, database credentials, etc for your Zen Cart dev site. If you're running in a Zen Cart Habitat environment, the *WEBTEST_ADMIN_PASSWORD_INSTALL* should be set to *developer1*.

### 1.6.8 Running Web Tests

a) Start Selenium engine from command line:

    java -jar <path_to>/selenium-server-standalone-<version>.jar -trustAllSSLCertificates

b) Start the webtests in another terminal window:

    phpunit -c testFramework/webtests/phpunit.xml

One of the joys of Open Source projects is the collective sharing of knowledge, skill, and creativity ... for the common good of everyone.

To that end, the documentation in this section is here to help you get started contributing to Open Source, specifically to the Zen Cart project.

CHAPTER 2

# Developer Documentation

## 2.1 Admin Classes

Currently classes that relate solely to admin are mainly stored in the *admin/includes/classes* directory.

In v157+ any new classes will be stored in the *catalog* classes directory under an admin directory.

e.g. *includes/classes/admin/*

> **Warning:** For now, we will not accept any pull requests that move current admin classes to the new directory. This is to ensure backward compatibilty of plugins. This may change in future Major releases.

## 2.2 FileSytem Class

The File System classs provides a number of methods for loading files from directories within Zen Cart and where necessary load from overrides and installed plugins.

The File System class is a Singleton and can be instantiated with,

```
$fileSystem = FileSystem::getInstance();
```

note also to use this class in your code, you will need to have a *use* statement.

```
use Zencart\FileSytem\FileSytem;
```

methods currently provided are :-

- loadFilesFromDirectory($rootDir, $fileRegx)
- listFilesFromDirectory($rootDir, $fileRegx)
- loadFilesFromPluginsDirectory($installedPlugins, $rootDir, $fileRegx)
- findPluginAdminPage($installedPlugins, $page)

## 2.3 Language Loader Class

The LanguageLoader class is responsible for loading language files, including overrides etc.

**Note:** Currently only have a languageLoader for admin, catalog coming soon.

**Note:** More specifics on order of loading for admin/catalog coming soon.

### 2.3.1 Admin Language Loader

The order of loading for Admin Language files is as follows.

**Todo:** Admin language loading order

**Warning:** This loading order may change in the future to match how loading works for catalog

## 2.4 Table View Controllers

*TableViewControllers* allow for the rapid development of admin pages that are mainly concerned with doing CRUD (Create, Read, Update, Delete) actions on a database table.

Currently creating a page of this type involves a lot of copy/pasting/customising an existing page.

TableViewControllers allow for easier building of this type of admin page/view. They also allow for much easier integion/overriding by plugins.

### 2.4.1 Page Skeleton

Pages that want to use the table view have a simple skeleton.

```
require('includes/application_top.php');

require_once DIR_FS_CATALOG . DIR_WS_CLASSES . 'QueryBuilder.php';
require_once DIR_FS_CATALOG . DIR_WS_ADMIN_CLASSES . 'TableViewControllers/
↪TableViewController.php';
require_once DIR_FS_CATALOG . DIR_WS_ADMIN_CLASSES . 'TableViewControllers/
↪BaseController.php';


$tableDefinition = [];

$tableController = (new YourTableController(
    $db, $messageStack, new QueryBuilder($db), $tableDefinition, ))->processRequest();


?>
```

(continues on next page)

(continued from previous page)

```
<!doctype html>
<html <?php echo HTML_PARAMS; ?>>
<head>
    <meta charset="<?php echo CHARSET; ?>">
    <title><?php echo TITLE; ?></title>
    <link rel="stylesheet" type="text/css" href="includes/stylesheet.css">
    <link rel="stylesheet" type="text/css" href="includes/cssjsmenuhover.css" media=
↪"all" id="hoverJS">
    <script src="includes/menu.js"></script>
    <script src="includes/general.js"></script>
    <script>
        function init() {
            cssjsmenu('navbar');
            if (document.getElementById) {
                var kill = document.getElementById('hoverJS');
                kill.disabled = true;
            }
        }
    </script>
</head>
<body onload="init()">
<!-- header //-->
<?php require(DIR_WS_INCLUDES . 'header.php'); ?>
<!-- header_eof //-->

<!-- body //-->

<?php require "includes/templates/table_view.php"; ?>

<!-- body_eof //-->

<!-- footer //-->
<?php require(DIR_WS_INCLUDES . 'footer.php'); ?>
<!-- footer_eof //-->
</body>
</html>
<?php require(DIR_WS_INCLUDES . 'application_bottom.php'); ?>
```

### 2.4.2 Table Controller Classes

### 2.4.3 Table Definition Array

The table definition array sets certain parameters that control the building of the table view.

e.g.

```
$tableDefinition = ['colKey'           => 'unique_key',
                    'actions'          => [['action' => 'new', 'text' => 'new',
↪'getParams' => [], 'showOnlyOnEmptyAction' => true]],
                    'defaultRowAction' => '',
                    'columns'          => ['unique_key' => ['title' => TABLE_HEADING_
↪KEY],
                                           'name'       => ['title' => TABLE_HEADING_
↪NAME],
                                           'status'     => ['title' => TABLE_HEADING_
↪STATUS]]
```

(continues on next page)

(continued from previous page)

- colKey Sets the get parameter for the colums that is the key for links
- actions Alows the setting of an array of *Action Buttons* for the table
- defaultRowAction Sets the default row action for the table
- columns Defines the columns that will be displayed in the table

### 2.4.4 colKey

The *colKey* parameter defines a get parameter ($_GET) name that is used as the index for each row in the table.

For example in the *countries table* this would be *countries_id* and so the entry would be :-

```
'colKey' => 'countries_id'
```

### 2.4.5 actions

Defines one or more action buttons that will be displayed below the table. For example you may want a *new* button to add a new entry.

Each action entry consists of a number of parameters.

#### 2.4.5.1 action

This represents the *action* that clicking he button will perform. e.g setting *action* to *new* will add *action=new* to the button link.

#### 2.4.5.2 text

This is the text that will be applied to the button.

#### 2.4.5.3 getParams

Additional get parameters that are added to the button link. Note: $_GET[‘page’] will automatically be set if it is already set in the URI. additionaly the *colKey* parameter will also be automaticaly set in the button URL as well.

#### 2.4.5.4 showOnlyOnEmptyAction

The button will only be shown if there is not an action get parameter. e.g. $_GET[‘action’] is not set

### 2.4.6 defaultRowAction

Normally when the currently selected row of the table is clicked, this will generate an edit action for that row resource. This parameter allows for altering that behavior.

```
'defaultRowAction' => ''
```

### 2.4.7 columns

Defines the columns that will be displayed for the table.

## 2.5 Plugin System

### 2.5.1 Directory Structure

Plugins written for the new achitecture will reside in the zc_plugins directory. Within their directory, the plugins can add directories that mimic the main Zen Cart directories. note how each plugin also has a versioned directory. This allows for automated upgrades and the possiblity of downgrading as well.

e.g.

- zc_plugins
  - rewardPoints
  - manifest.php
    - v1.0.0
    - manifest.php
      - admin
      - includes
      - auto_loaders
      - init_includes
      - extra_datafiles
      - classes
      - languages
      - catalog
      - includes
      - functions
      - extra_functions
      - templates
      - etc
    - v1.0.1
    - manifest.php
      - admin
      - includes
      - auto_loaders
      - init_includes
      - extra_datafiles
      - classes

- languages
- catalog
- includes
- functions
- extra_functions
- templates
- etc

### 2.5.2 Manifest Files

Each plugin will have a master manifest.php in it's main directory and version specific manifest files in each version directory.

The main manifest file looks like :-

Listing 1: main plugin manifest.php

```
<?php
return [
    'pluginType' => 'free',
    'pluginAuthor' => 'plugin author',
    'pluginName' => 'Reward Points',
    'pluginDescription' => 'Reward Points Description',
    'managed' => true,
    'zcVersions' => ['v157'],
    'pluginGroups' => []
];
```

while version specific manifests look like

Listing 2: plugin version manifest.php

```
<?php
return [

    'pluginAuthor' => 'plugin author',
    'pluginVersion' => 'v1.0.0',
    'zcVersions' => ['v157'],
];
```

#### 2.5.2.1 pluginType

Currently we allow only two type, either *free* or *core*. All 3rd party plugins should be marked as *free*

#### 2.5.2.2 pluginAuthor

A human readable string for the plugin author.

#### 2.5.2.3 pluginName

A human readable string for the name of the plugin.

#### 2.5.2.4 pluginDescription

A human readable string describing the plugin.

#### 2.5.2.5 managed

All plugins written to the new v157 spefication should have *managed* => *true*. Legacy plugins can add a manifest file where *managed* => *false* to allow for listings of plugins installed.

#### 2.5.2.6 zcVersions

An array of the Zen Cart versions the plugin supports.

#### 2.5.2.7 pluginGroups

*not surrently supported* but will eventually support filtering in admin.

#### 2.5.2.8 @todo

manifest for file hashes to allow for security and auto upgrades etc.

### 2.5.3 Installer Language Files

While most language constants for the plugin installer are managed internally there will be cases where a plugin needs to define some of it's own language defines.

For example, if the plugin needs to test some pre-requisites befor installing, the error messages when those pre-requisites fail will need to be defined.

In these cases the plugin system allows for loading custom language files.

The main custom language file would reside in the *plugin version directory*/languages/*language name*/main .php file.

e.g.

- zc_plugins
  - rewardPoints
    - v1.0.0
      - Installer
      - languages
      - english
      - *main.php*

If the plugin needs more customisation and wants to separate out other language defines into separate files it may also load the language files separately.

A helper method in the installer class can be used to do this.

e.g.

```
$this->loadInstallerLanguageFile('myFile.php');
```

**Note:** Rememeber, we are talking here about language defines that are only used during installation. Loading language files needed by the plugin itself are handled differently.

### 2.5.4 Installer Classes

For most cases the base installer class can be used without customisation to install a plugin.

However there may be times when a plugin needs to override the default installer behavior.

For example a plugin may need to check some pre-requisites before allowing installation.

This may be the availability of a php extension or maybe the plugin relies on another plugin being installed.

In these cases the plugin can define it's own installer class that extends the base installer.

The class must be named *PluginInstaller.php* and be placed in the *Installer* directory.

e.g.

- zc_plugins
  - rewardPoints
    - v1.0.0
      - Installer
      - *PluginInstaller.php*

The skeleton of this class should look like :-

```
<?php
class PluginInstaller extends BasePluginInstaller
{
}
```

There are a number of methods that the installer class can override.

- prerequisitesCheck

### 2.5.5 SQL Installation

Plugins may need to create/alter dtabase tables and/or insert/amend data in database tables.

The plugin installer allows two methods for doing this. Using .sql files containing just sql statements or using a class based migration system.

#### 2.5.5.1 Plain SQL Files

Listing 3: Example

```
CREATE TABLE IF NOT EXISTS reward_master (
                           rewards_products_id INT( 11 ) NOT NULL AUTO_INCREMENT␣
↪PRIMARY KEY,
                           scope INT( 1 ) NOT NULL DEFAULT '0',
                           scope_id INT( 11 ) NOT NULL DEFAULT '0',
                           point_ratio DOUBLE( 15, 4 ) NOT NULL DEFAULT '1',
                           bonus_points DOUBLE( 15, 4 ) NULL,
                           redeem_ratio DOUBLE( 15, 4 ) NULL,
                           redeem_points DOUBLE( 15, 4 ) NULL,
                           UNIQUE unique_id ( scope , scope_id ));

INSERT INTO reward_master
(rewards_products_id ,scope ,scope_id ,point_ratio ,bonus_points, redeem_ratio,␣
↪redeem_points)
VALUES (NULL , '0', '0', '1.0000', NULL, 0.01, NULL);
```

The sql file should reside in

- zc_plugins
  - rewardPoints
    - v1.0.0
      - Installer
      - sqlInstall
      - install.sql

**Warning:** As Zen Cart currently uses mainly MyIsam tables, there is no way to safely roll back installer sql if an error occurs. Some support for rollback may be added later (using generated migrations).

#### 2.5.5.2 Migration Classes

**Todo:** This is a a todo

### 2.5.6 Writing Plugins

As mentioned earlier, the directory structure for plugins somewhat resembles the normal Zen Cart directories.

As such some files in certain directories will autoload automatically, and other s may need to be loaded manually.

See the sections below for an understanding of where to place files and how those files will be loaded.

#### 2.5.6.1 PSR 4 Autoloading

Autoloading is provided for Plugins. Two primary namespaces are created for installed plugins.

Zencart\Plugins\PluginKey\Admin

Zencart\Plugins\PluginKey\Catalog

where *PluginKey* is the *unique key* for the plugin with the first charachter upper cased.

You can also use your own namespacing convention for your plugins classes. However as we cannot know which version of your plugin is installed at runtime, a helper function to get the correct directory is provided.

e.g.

```
$path = $pluginManager->getPluginDirectory($pluginName, $installedPlugins);
```

where *$pluginName* is the *unique key* for your plugin and *$installedPlugins* is a list of installed plugins. This variable is provided in the global namespace and therefore availbale throughout the *initSystem*.

$path will point to the root directory of your plugin, deoendant on the version installed.

e.g.

*zc_plugins/rewardPoints/v1.0.1/*;

You can then register the namespce using :-

```
$psr4Autoloader->addPrefix('Author\Plugin', $path . 'some_directory_path' );
```

again *$psr4Autoload* is a global variable availaboe throughout the *initSystem*

#### 2.5.6.2 Admin Controllers

To allow for the loading of pages/views in admin, we can't use the old URI structure of

*https://mydomain.com/admindir/somepage.php* as there is then no way to tell if the page is a core page or a plugin page.

To allow plugin pages to be auto-detected and loaded the URI structure for admin has been changed to a similar structure as catalog pages.

e.g.

*https://mydomain.com/admindir?cmd=somepage*

The zen_href_link function in admin has been updated to reflect this.

#### 2.5.6.3 InitSystem

The *initSystem* file auto loader and init_includes can be used within plugins and will load automatically.

i.e. files in the plugin *admin/includes/auto_loader* and *admin/includes/init_includes* direcories will function exactly as they do in a normal Zen Cart install.

No adjustment needs to be made to file paths used in the *initSytem* auto_loader, as this will be handled by the plugin infrastructure, inluding ignoring loading if the plugin is not installed.

---

**Note:** Please see the *InitSystem* for more information on how the *initSystem* works.

---

#### 2.5.6.4 Support Files

Support files include files that exist within the *includesextra_datafiles* and *includes/functions/extra_datafiles* directories. Again all of these files will be automatically included from the releavant plugin directories.

**Warning:** files within *includes/extra_configures* are not part of the plugin structure.

#### 2.5.6.5 Page Files

Standard page files in either the plugin admin or catalog directory will load automatically as well as pulling in language files related to thet page/view.

#### 2.5.6.6 Language Files

@todo

#### 2.5.6.7 Template Files

@todo

### 2.5.7 Debugging Plugins

### 2.5.8 Plugin Manager Class

The PluginManager class provides support for installing/upgrading plugins and querying a plugins installed and other plugin statuses.

#### 2.5.8.1 Plugin Manager Public Methods

While we had started a plugin architecture for the V2 Develop branch, there was naturally a lot of concern over the fact that current plugins would no longer work and would need extensive rewriting.

Now it is also the case that plugins will also need modification for the v156 and later branches, although in the main this mainly relates to the new template/css for admin along with some other minor tweeks.

Many plugins do not take advantage of a lot of the new notifiers added to v15x which in some cases mean they need to overrite/amend Zen Cart core files, and for a more automated plugin system this will need to change.

As well, installation of plugins is labour intensive on the part of shop owners/Developers, where files from the plugins need to be moved to multiple directories within the Zen Cart framework.

Given this it is intended to introduce a new plugin infrastructure into the v157 branch.

However it should be noted that we are not going to force plugin authors to rewrite plugins to conform to this new plugin framework.

Plugins that work with the current code will continue to work.

## 2.6 InitSystem

CHAPTER 3

# Habitat Developer Environment

**Note:** Habitat as a development environment is no longer being updated. You may want to try using a docker based environment instead.

Habitat is a Vagrant based virtual machine, with the ability to custom-provision the VM automatically based on settings you decide.

This is great for use on a laptop for portable offline use, or for doing development testing without installing special versions of PHP/MySQL/Apache/etc.

With default settings, it will link a current checkout of the zencart github repository to a domain name that you can access directly from your browser, all from within your own computer.

A yaml-based configuration system lets you customize your system by adding new shared directories which may map to new websites, and automatically clone other github repositories/branches into those websites.

See the links to the left for detailed instructions for installation and customization.

Habitat was inspired by Laravel Homestead

You can contribute to this documentation by forking the zencart/documentation repository on github and submitting Pull Requests.

## 3.1 Installation

### 3.1.1 Prerequisites

Install Virtualbox and Vagrant > *Ubuntu* > sudo apt-get install virtualbox sudo apt-get install vagrant > *Windows or Mac OSX* > https://www.virtualbox.org/wiki/Downloads https://www.vagrantup.com/downloads.html

### 3.1.2 Install Habitat

Clone the habitat repository. > *Command Line* > git clone https://github.com/zencart/habitat

Then change into the newly created habitat directory.

OPTIONAL: Add the Virtual Machine to Vagrant. (You don't have to do this manually. It will happen automatically for you.) > *Command Line* vagrant box add zencart/habitat https://s3.amazonaws.com/zencart-vagrant-boxes/habitat.box

Update your hosts file > *Ubuntu or Mac OSX* > /etc/hosts > *Windows* > %systemroot%system32driversetchosts Where %systemroot% is usually c:/windows

And add this line > 172.22.22.22 zen.local 172.22.22.22 devdocs.local

Before you start up the VM you may want to customise the environment. If so see the [Customising](customising.md) page before starting the VM.

To start the VM: > *Command Line* > vagrant up

## 3.2 Code Development

The Habitat VM initially links a domain, *zen.local*, to a checkout of the Zen Cart® develop branch, unless you have customised the Habitat.local.yaml.

You should be able to just open a browser and enter the http://zen.local URL to see the site. The directory that the code is stored in within the VM is shared to the habitat directory that was created earlier under the habitat/web/zen folder.

As this directory is shared with the virtual machine, you can edit the files locally in your favorite editor, and see the changes immediately reflected in your browser.

More information on managing your VM can be found on the Vagrant website.

### 3.2.1 Passwords

```
The user/password for MySQL is
user = zencart
password = zencart
The root password for MySQL is also zencart
```

## 3.3 Customising Habitat

When Habitat first creates the development environment, it looks for a file called Habitat.yaml, which it will use to provision you Virtual Machine.

If *Habitat.yaml* does not exist, it will copy the included *Habitat.yaml.example* to *Habitat.yaml*

By default the example file will do a checkout of the latest develop branch from Github. If you want that checkout to say be your forked branch, then you will need to create the Habitat.yaml first and adjust it to point to your own forked repo instead of the github.com/zencart repo.

**Note:** If you make changes to the yaml file after doing *Vagrant up*, you will need to run . . .

*Command Line*

vagrant provision

in order for the changes to take affect.

### 3.3.1 Habitat.yaml Configuration Options

The configuration options for the YAML file are as follows. NOTE: Some of these are NOT shown in the example file.

**hostname:**

This defaults to *habitat*. If you want to rename the machine to differentiate multiple instances (as they show up in VirtualBox), you can set this here.

Normally best left as-is, or skipped altogether.

**ip:**

This is the ip that is assigned to your virtual machine. You should only change it if the current IP clashes with settings on your local machine/network If assigning a new IP address, you should not use a publicly available IP, but an IP from one of the private IP spaces.

The default is *172.22.22.22*

**memory:**

The amount of memory assigned to your virtual machine.

The default is *1024*

**cpus:**

The number of cpu cores assigned to your virtual machine.

The default is *1*

**sites:**

This option can be used create other virtual domains that map to a directory and optionally clone a github repository into that directory

By default it contains:

**sites:**

- domain: zen.local dir: develop skeleton: apache.default git_url: https://github.com/zencart/zencart.git branch: develop
- domain: devdocs.local dir: devdocs skeleton: apache.default git_url: https://github.com/zencart/documentation.git branch: master

If you want to add another site, you can add something like:

- domain: v154.local dir: v154 skeleton: apache.default git_url: https://github.com/zencart/documentation.git branch: v154

The options within the *sites:* directive are:

**domain:**

This is the name of the domain that you want to map to your shared site. You will also have to create an entry in your *hosts* file for this same domain.

**dir:**

This is the directory, relative to the shared Habitat/web directory, that you want to be the web root of your site.

See *do_default_map:* and *to:* below for more info.

**skeleton:**

In order to provision apache correctly, we use a skeleton apache vhost file.

These skeleton files are stored in the shared scripts/skeletons directory.

Currently only the *apache.default* skeleton is used.

**git_url:**

This directive is optional, but if set will be used as the repository to clone. It is typical to use a github.com or bitbucket.org URL here, for example: *https://github.com/zencart/zencart.git*

**branch:**

This directive is optional, but if set will be used as the branch to checkout, in reference to the *git_url* directive.

**docroot:**

This directive is optional, but if you need to override or set a different document-root (ie: re-point to a subdir like 'public') set that by adding this directive. For example: *docroot: /public*

**do_default_map:**

This controls whether to share the internal Habitat directory with the host machine.

The default is true. There is little reason to change this setting unless you have advanced knowledge of managing a vagrant virtual machine.

**folders:**

The default mapping is to map the *map:* folder from your host PC to the *to:* folder in your VM. If *create:* is set to *true* then it will also create the folder on both ends if it doesn't already exist.

**map:**

The local PC folder where files are stored on your PC. These files will be synced into the *to:* folder of the vm, and vice-versa.

The default is assumed to be the local folder from which you're running vagrant/habitat.

**to:**

This is the name of the folder inside vagrant/habitat which the files from the *map:* folder will be synced to/from the vm.

The default is */home/vagrant/web*, and the *dir:* directive will be put inside this folder.

**create:**

This determines whether to create the folder automatically if it doesn't already exist.

The default is true.

**authorize:**

This can be used to specify which RSA public key file you want copied to the vm for easy login from your host machine's command-line if you don't want to use the default *vagrant ssh* command. See the vagrantup.com documentation for more detail.

**keys:**

This is a list of private RSA keys you want copied to the VM. This can be useful if you're having any difficulty connecting to github or bitbucket due to private-key problems.

**localize_tools:**

This determines whether to map the vm's */tools* folder to your host machine. This is required (useful) to run the supplied unittesting tools. See the [Tools](dev-tools.md) documentation.

The default is *true*

## 3.4 Development Tools

### 3.4.1 phpMyAdmin

phpMyAdmin is pre-installed on the vm, and can be easily accessed via *http://172.22.22.22/phpmyadmin* and logging in as zencart/zencart.

###TIP: >For easy importing of .sql files, note that on your host PC there is a *habitat/tmp* folder. This is shared with phpMyAdmin and makes quick importing of .sql files very simple.

>1. Simply copy any desired .sql files to your host PC's *habitat/tmp* folder 2. Open phpmyadmin in your browser, *select a database*, and then click the *Import* button. There will be a dropdown available with a list of all the .sql files above.

### 3.4.2 Database Backup and Restore

**BACKUP**

To do a complete backup of all databases in the vm:

1. ssh into the habitat vm, so you have a command-line prompt
2. cd into the folder that you've shared back to your host machine
3. Enter this command to tell MySQL to dump all databases to a single gzipped file using the current date/time stamp as the filename:

> *Command Line* > mysqldump -u zencart -pzencart –all-databases –hex-blob | gzip -9 > ./habitat_$(date +%Y-%0m-%0d_at_%0I_%0M_%0p.sql.gz)

**RESTORE**

Similarly, to restore:

1. Put the "backup" file into your shared folder
2. Unzip it so the file is a .sql file with text content (not zipped)
3. ssh into the habitat vm
4. cd into the shared folder
5. Use the following command to initiate the restore, replacing HABITAT_FILENAME with the actual filename of your .sql file (ie: *habitat_YYYY-MM-DD_at_HH_MM.sql*):

> *Command Line* > mysql -u zencart -pzencart < HABITAT_FILENAME

### 3.4.3 Unit Testing

Habitat includes a helper script to help with running unit tests within the Zen Cart testFramework. This depends on having *localize_tools:* and *do_default_map:* set to true in your yaml file. It may encounter some difficulties if the *folders:map* or *folders:to* settings have been customized.

From the command line you can do > *Command Line* > cd habitat/tools ./unittest DIRNAME

The option given to the unittest script is a reference to a site in your habitat VM. In other words it should match the *dir* option given in a *sites:* block of your yaml file. To run the tests on the *develop* branch, on your host machine you'd use *./unittest develop*

### 3.4.4 phpDocumentor

Habitat also includes a helper script to generate phpDocumentor output, which will be placed in the *DIRNAME/docs/phpdocs* folder.

Syntax is similar to the unit-testing above, and depends on the same prerequisites.

> *Command Line* > cd habitat/tools ./generatedocs DIRNAME

## 3.5 Technical Specifications

### 3.5.1 Software Installed in Habitat

- Ubuntu 14.04.2 LTS trusty64 x86
- Apache 2.4.12 with SSL along with the following modules
  - mod_access_compat
  - mod_alias
  - mod_auth_basic
  - mod_authn_core
  - mod_authn_file
  - mod_authz_core
  - mod_authz_host
  - mod_authz_user
  - mod_autoindex
  - mod_deflate
  - mod_dir
  - mod_env
  - mod_filter
  - mod_mime
  - mod_mpm_prefork
  - mod_negotiation
  - mod_php5
  - mod_rewrite
  - mod_setenvif
  - mod_socache_shmcb
  - mod_ssl
  - mod_status

- MySQL 5.5.43
- phpMyAdmin 4.0.10
- **PHP 5.6.10, with customized PHP configurations:**
  - *error_reporting* = E_ALL
  - *display_errors* = On
  - *memory_limit* = 512M
  - *date.timezone* = UTC
  - *post_max_size* = 512M
  - *upload_max_filesize* = 512M
  - *html_errors* = Off
  - *error_log* = *homevagranthabitatphp_errors.log* (Thus the *error_log* is mapped to the host machine for easy reference)
- git 1.9.1
- Composer 1.0-dev 2015-06-26

  _You can update *composer* to the latest version from the Habitat console (via *vagrant ssh*) by running *composer self-update*
- Other tools installed via Composer
  - phpunit
  - phpmd
  - php_codesniffer
  - php-coveralls
  - vfsStream
  - phpdocumentor

### 3.5.2 Updating

From time to time updates will be published for the Habitat VM.

But in the meantime if you need to apply updates, use *vagrant ssh* and run *sudo apt-get update && apt-get upgrade*

### 3.5.3 Source

The Habitat Virtual Machine is built using zencart/pioneer

CHAPTER 4

# Docker Development Environment

While there is no official Docker image for working with Zen Cart, we can use other projects to develop Zen Cart with Docker.

This documentation will explain how to use Devilbox to achieve this.

## 4.1 Installing Docker

Before installing Devilbox, we first need to install the docker daemon and docker-compose

- Windows
- Mac
- Linux

## 4.2 Installing Devilbox

### 4.2.1 Customising Devilbox