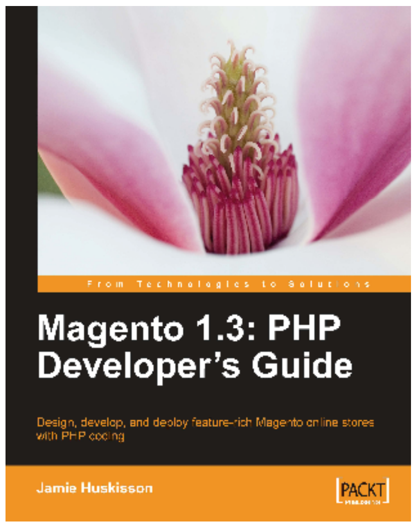
Magento 1.3: PHP Developer's Guide


From Technologies to Solutions
Magento 1.3: PHP Developer's Guide
Design, develop, and deploy feature-rich Magento online stores with PHP coding
Jamie Huskisson
PACKT

Jamie Huskisson

Chapter No. 3
"Magento's Architecture"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.3 " Magento's Architecture "

A synopsis of the book’s content

Information on where to buy this book

About the Author

Jamie Huskisson, a passionate 23-year-old freelance developer from Nottingham, has been working with Magento for the past two years since the very early BETA versions. His development client list features names such as NHS, Volkswagen, and Nike with his day-to-day work life spent building everything from web applications to e-commerce stores and small business sites. He also trains groups of developers, and provides consulting on adopting open source technologies over closed systems for clients when required.

Jamie also writes and maintains a popular online blog at `http://www.jhuskisson.com/` where he gives advice on various aspects of the web, including freelancing, Magento, Wordpress, PHP, and running your own business.

I'd like to thank my girlfriend Vicky for putting up with my late nights working on the book. I'd also like to thank my family and especially my mother, for raising me to believe that I can achieve anything I put my mind to.

To everyone that reads this, enjoy your time developing what you read in and out of this book. I look forward to hearing from any of you that develop sites or modules based on what you read between these covers.

Magento 1.3: PHP Developer's Guide

Magento 1.3: PHP Developer's Guide will guide you through development with Magento, an open source e-commerce platform. Exploring commonly approached areas of Magento development, Magento 1.3: PHP Developer's Guide provides you with all the information you'll need to get a very solid understanding of developing with Magento.

What This Book Covers

Chapter 1, *Magento 3.1: PHP Developer's Guide* shows you what this book will cover entirely in detail for you to read through.

Chapter 2, *Installing/Upgrading Magento and Preparing for Development* will prepare you for development with Magento as well as showing you how to install and upgrade Magento using a variety of different methods.

Chapter 3, *Magento's Architecture* introduces you to Magento's architecture, the Zend framework, and how the system works from a development point of view.

Chapter 4, *Shipping Modules in Magento* shows you how to put together shipping modules in Magento to handle shipping calculation and information.

Chapter 5, *Building a Payment Module for Magento* guides you in putting together payment methods in Magento and building connecting modules between Magento and the payment gateway of your choice.

Chapter 6, *Building a Basic Featured Products Module* walks you through building a featured product module into your web site so that you can show featured products in your Magento categories.

Chapter 7, *Fully-Featured Module for Magento with Admin Panel* shows you how to put together a fully featured module in Magento as well as giving it a full backend to manage data with. You'll also learn how to use the module creator to quickly deploy module skeletons to use yourself in the future.

Chapter 8, *Integration of Third-Party CMS* will show you how to integrate Wordpress with your Magento installation. It will also show you the other options available should you use any other content management systems.

Chapter 9, *Magento's Core API* guides you through the Magento Core API and how to utilize it with scripts of your own to interface with Magento's data.

Chapter 10, *Importing and Exporting Data* shows you how to work with import and export data profi les in Magento to work with basic order, product, and customer data.

Appendix, *Resources for Further Learning*, contains additional resources for further learning. Its not a part of this book and it can be downloaded from Packt's website `//www.packtpub.com/files/7429-Appendix-Resouces-for-Further -Learning. pdf`.

3
Magento's Architecture

Magento has a wonderful architecture behind its system. It's a very strict architecture that relies on us knowing where the files should be placed and how to structure our templates and modules. But this is part of what makes Magento a great system, in that it enforces these standards.

Here in this chapter, we will learn about this architecture and how it applies to development with Magento. We will learn:

- Where everything is within Magento
- What all the base directory files and folders do
- The basics of how the template system works
- How modules work within the system
- How the Zend Framework fits into the equation
- The best methods for backing up Magento

Magento's base structure

The fundamental knowledge of Magento's architecture begins with its file structure. It's important to know what goes where by default, so that we may position our new files accordingly, especially in terms of ensuring that our development doesn't overwrite core files.

Base directory

The default installation contains the following files and directories in the base directory:

- `.htaccess`
- `.htaccess.sample`
- `404` (directory)
- `app` (directory)
- `cron.php`
- `downloader` (directory)
- `favicon.ico`
- `index.php`
- `index.php.sample`
- `js` (directory)
- `lib` (directory)
- `LICENSE_AFL.txt`
- `LICENSE.txt`
- `media` (directory)
- `pear`
- `pkginfo` (directory)
- `report` (directory)
- `skin` (directory)
- `var` (directory)

Each of these files and directories has a different purpose. We'll go through them to ensure that we understand the function of each. This will help us later, if ever we need to find something specific, or when developing. It will also be helpful when we'll be looking to place the files coming out of our new module into the appropriate directory.

The function of each of the files in the base directory

The following is a run through of all the files in the base directory, to show us what they do:

- `.htaccess`—This file controls `mod_rewrite` for fancy URLs and sets configuration server variables (such as memory limit) and PHP maximum execution time, so that Magento can run better.
- `.htaccess.sample`—Works as a backup for `.htaccess`, so that we know the default `.htaccess` file (if ever we edit it and need to backtrack).
- `cron.php`—The file that should be executed as a `cron` job every few minutes to ensure that Magento's wide caching doesn't affect our server's performance.
- `favicon.ico`—Magento's default `favicon`; it's the small icon that appears in the toolbar of our browser.
- `index.php`—The main loader file for Magento and the file that initializes everything.
- `index.php.sample`—The base template for new `index.php` files, useful when we have edited the `index.php` file and need to backtrack.
- `LICENSE_AFL.txt`—It contains the Academic Free License that Magento is distributed under.
- `LICENSE.txt`—It contains the Open Software License that Magento is distributed under.
- `pear`—This controls all automatic updating via the downloader and SSH. This file is initialized and handles the updating of each individual module that makes up Magento.
- `php.ini`—A sample `php.ini` file for raw PHP server variables recommended when setting up Magento on our server. This should not be used as a complete replacement, but only as a guide to replace certain lines of the `php.ini` server file. It is useful when overriding these variables when `.htaccess` isn't enabled on our server.

The function of each of the folders in the base directory

The following is a run through of all the folders in the base directory to show us their contents:

- `404`—The default `404` template and skin storage folder for Magento.
- `app`—All code (modules), design (themes), configuration, and translation files are stored in this directory. This is the folder that we'll be working in extensively, when developing a Magento powered website. Also contained in this folder are the template files for the default administration theme and installation.
- `downloader`—The web downloader for upgrading and installing Magento without the use of SSH (covered in Chapter 2).
- `js`—The core folder where all JavaScript code included with the installation of Magento is kept. We will find all pre-compiled libraries of JavaScript here.
- `lib`—All PHP libraries used to put together Magento. This is the core code of Magento that ties everything together. The Zend Framework is also stored within this directory.
- `media`—All media is stored here. Primarily for images out of the box, this is where all generated thumbnails and uploaded product images will be stored. It is also the container for importing images, when using the mass `import/export` tools (that we'll go through in Chapter 10).
- `pkginfo`—Short form of package information, this directory contains text files that largely operate as debug files to inform us about changes when modules are upgraded in any way.
- `report`—The skin folder for the reports that Magento outputs when any error occurs.
- `skin`—All assets for themes are stored within this directory. We typically find images, JavaScript files, CSS files, and Flash files relating to themes, in this directory. However, it can be used to store any assets associated with a theme. It also contains the skin files for the installation of skins and administration templates.
- `var`—Typically where we will find all cache and generated files for Magento. We can find the cache, sessions (if storing as files), data exports, database backups, and cached error reports in this folder.

The template system architecture

The template architecture is broken into three areas—two for development of the theme and one for the containment of the assets:

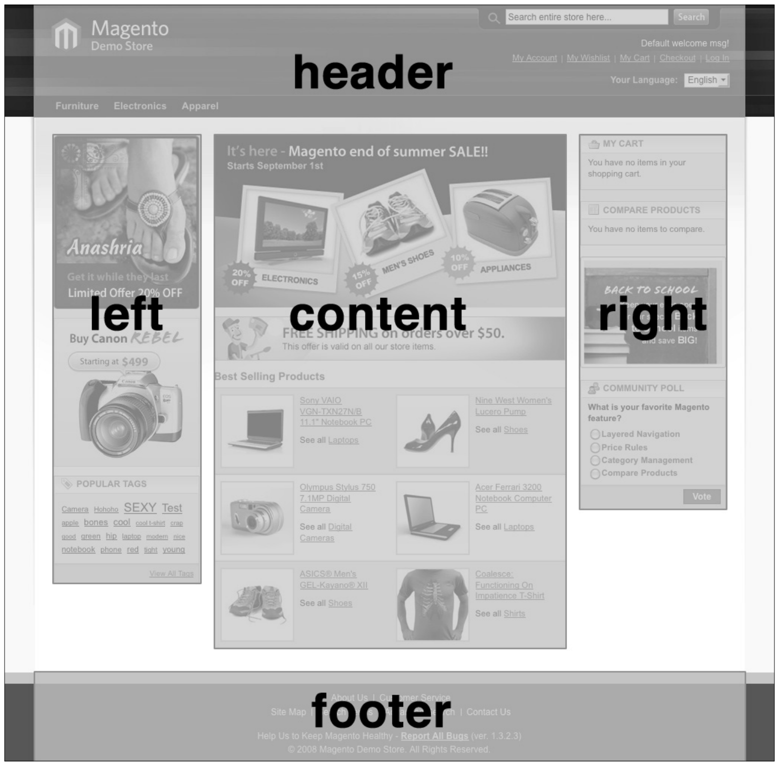
- `/app/design/frontend/default/<template_name>/`
    - `layout/`—For all the XML files declaring which module tied functions should be called to which template files
    - `template/`—For all the templates processing the output that is passed from functions called from `layout/` and structured into the final output to the user.
- `/skin/frontend/default/<template_name>/`—For the containment of all assets relating to our template, images, CSS, Flash, and JavaScript.

Structural blocks and content blocks

Each theme contains structural and content blocks. Structural blocks are the ones that lay out the theme into sections. Let's take a look at a three-column layout. The following are the structural blocks in a three-column layout:

- **header**
- **left**
- **content**
- **right**
- **footer**

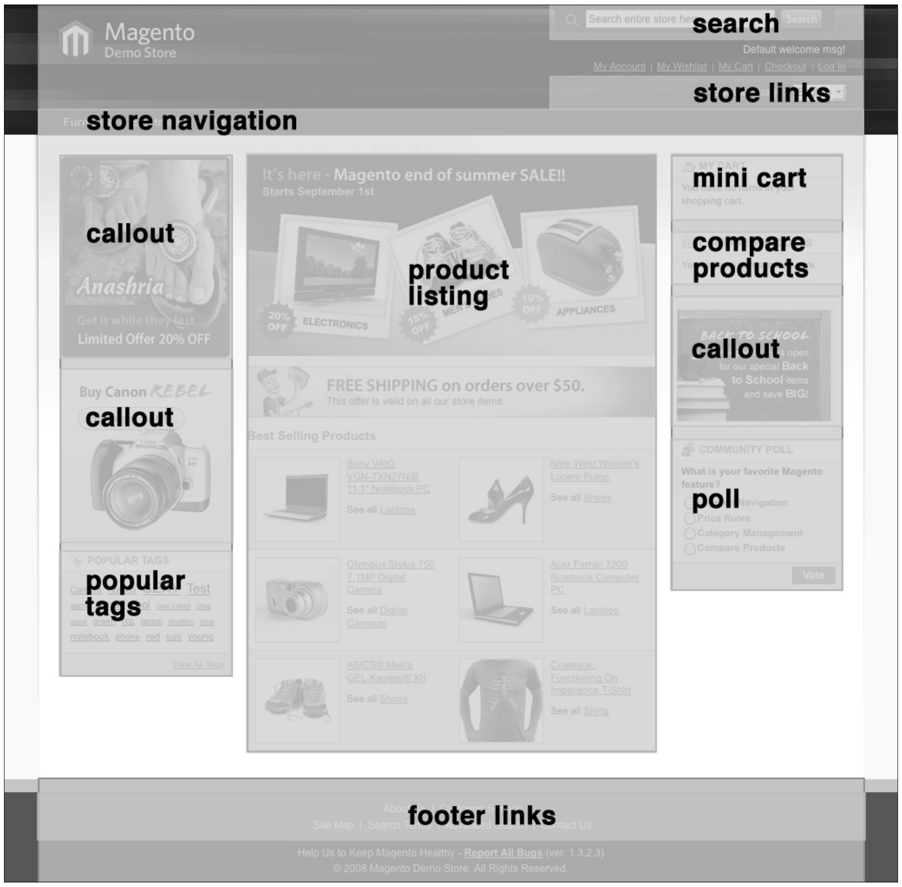
Here's a visual representation of those structural blocks laid over the Magento demo store:



In each of the structural blocks, we then have content blocks that give each structural block its content for output to the browser. Let's take the **right** column; our content blocks set for this column on a standard theme could be:

- **mini cart**
- recently viewed products
- newsletter subscription block
- **poll**

Here we have a visual representation of these content blocks on top of the Magento demo store:



On receiving a request from a user connecting to our site to view the page:

1. Magento will load the structural areas
2. Each structural area will be processed through
3. Magento will gather the content blocks assigned to each structural area
4. It will then progress through the content block template for each structural area, to process the output
5. It sends all of this back as final output to the user, who then views the Magento page that was requested

XML layout files

To assign blocks to each of these structural blocks, Magento loads an XML layout file for each request. This XML layout file is called by the URL that the user is accessing on the site. It declares all modules that are to be loaded in each structural area of the site. On top of this, we have a `page.xml` file, which is the default loader for all pages on the site.

A layout XML file is typically structures as follows:

```
<default>
  <reference name="header">
    <block type="page/html_header" name="header" as="header">
      <block type="page/template_links" name="top.links"
             as="topLinks"/>
      <block type="page/switch" name="store_language"
             as="store_language"
             template="page/switch/languages.phtml"/>
      <block type="core/text_list" name="top.menu" as="topMenu"/>
    </block>
  </reference>
</default>
```

In the above code, we have:

- `<default>`—The handler for the URL, in this case default will load no matter what other handler is being initialized
- `<reference>`—The reference structure which calls the blocks in our theme
- `<block>`—A content block which defines the type of block and the template which will process the block's outgoing data in the system

In addition to this, Magento uses actions within blocks for functions which need to process the data that is input to them, for example adding CSS stylesheets:

```
<block type="page/html_head" name="head" as="head">
  <action method="addCss">
    <stylesheet> css/menu.css </stylesheet>
  </action>
  <action method="addCss">
    <stylesheet> css/clears.css </stylesheet>
  </action>
```

```
<action method="addItem">
  <type>js</type>
  <name>varien/iehover-fix.js</name>
  <params/>
  <if>lt IE 7</if>
</action>
<action method="addCss">
  <stylesheet>css/print.css</stylesheet>
  <params>media="print"</params>
</action>
  <action method="addCss">
    <stylesheet> css/print.css </stylesheet>
    <params> media="print" </params>
  </action>
</block>
```

We'll notice that there are several tags within the `action method` tag. These are processed into an array and then passed through the `action method=""` parameter, in this case `addCss`. This function then places the input into an output, ready for its appropriate template.

> Layouts are fully explained online in Magento's designer guide: `http://www.magentocommerce.com/design_guide/articles/intro-to-layouts`.

Hierarchical file processing

When creating new themes, we do not have to worry about copying all the theme and skin files from the default theme over to our new one. Let's presume that we have an additional theme called `new_theme`, alongside our `default` theme. Our theme calls files called `logo.gif` and `image.gif` on one of its pages.

The themes that we have contain the following files in their skin's images directory:

default	new_theme
`logo.gif`	`logo.gif`
`image.gif`	
`test.gif`	

Magento would process this main requesting `logo.gif` and `image.gif`. As `new_theme` is our current active theme, it will pull `logo.gif` from there., However, as `image.gif` does not exist in `new_theme`, Magento would grab that from `default`. So now, it works like this:

Requested file	Theme it will come from
`logo.gif`	`new_theme`
`image.gif`	`default`

Similarly, if `test.gif` were called in our template then it would come from the default theme. If we upload an image called `test.gif` to the image directory of `new_theme`, then it would immediately come from there instead.

This applies to all files for themes in Magento, which include the following:

- Templates
- Layout XML files
- Anything in the theme skin folders

Magento's template architecture and hierarchy is also explained online in the designer's guide to Magento: `http://www.magentocommerce.com/design_guide`

Modules and how they work within the system

Magento primarily works on a base of modules. All functionality is divided up into modules that make up the system overall. It's important to understand what each module does and how to go about adding modules to the system, in order to understand the architecture of modules themselves.

Distribution of the modules between directories

All modules are located within the `/app/code/` directory. Directories are commonly referred to as `codePools`. There are three possible locations for all modules that relate to the system. They are all split by type to prevent any confusion:

- `community`—For community-distributed extensions, usually those that we have installed through Magento Connect or have downloaded from a source, other than our own. Anything installed through Magento Connect will be installed here automatically.
- `core`—Reserved for core Magento modules, so that we cannot directly overwrite or interfere with them. We keep our modules out of core to avoid any conflict with the core modules or any future updates. Anything from a Magento upgrade or any new Magento modules will go into this directory.
- `Local`—This is where we should be placing our modules when they are either under local development or are not distributed among the community. It's best to keep anything that we develop in this directory, so as to not interfere with the core or community modules. Nothing will be automatically installed here, unless we have physically uploaded it.

Modules included with Magento

Included modules in the core folder of default Magento installation are as follows:

- `Mage_Admin`
- `Mage_AdminNotification`
- `Mage_Api`
- `Mage_Backup`
- `Mage_Bundle`
- `Mage_Catalog`
- `Mage_CatalogIndex`
- `Mage_CatalogInventory`
- `Mage_CatalogRule`
- `Mage_CatalogSearch`
- `Mage_Checkout`
- `Mage_Cms`
- `Mage_Contacts`
- `Mage_Core`

- `Mage_Cron`
- `Mage_Customer`
- `Mage_Dataflow`
- `Mage_Directory`
- `Mage_Downloadable`
- `Mage_Eav`
- `Mage_GiftMessage`
- `Mage_GoogleAnalytics`
- `Mage_GoogleBase`
- `Mage_GoogleCheckout`
- `Mage_GoogleOptimizer`
- `Mage_Install`
- `Mage_Log`
- `Mage_Media`
- `Mage_Newsletter`
- `Mage_Page`
- `Mage_Paygate`
- `Mage_Payment`
- `Mage_Paypal`
- `Mage_PaypalUk`
- `Mage_Poll`
- `Mage_ProductAlert`
- `Mage_Rating`
- `Mage_Reports`
- `Mage_Review`
- `Mage_Rss`
- `Mage_Rule`
- `Mage_Sales`
- `Mage_SalesRule`
- `Mage_Sendfriend`
- `Mage_Shipping`
- `Mage_Sitemap`
- `Mage_Tag`
- `Mage_Tax`

- `Mage_Usa`
- `Mage_Weee`
- `Mage_Wishlist`

Setting up the folder structure of a module

Let's presume that we want to set up a module's folder structure, ready for development. Our module's core folders will be placed in `/app/code/local/Book/Example/`.

These folders will primarily be used for storing our code that makes the module work. The folder structure breaks down as follows:

- `Block/`
- `controllers/`
- `etc/`
- `Model/`
    - `Mysql4/`
    - `Book/`
- `sql/`
    - `book_setup/`

Typically, developers will pick or choose each folder, depending on whether or not they're going to use it within their module.

Note that `Model/Mysql4/Book/` has its first letter in uppercase, whereas `sql/book_setup/` does not. We must be sure to keep this the same way throughout our development.

Template files for the frontend of our module will be stored as follows:

- XML files will be stored in `/app/design/frontend/<interface>/<theme>/layout/example/`
- Output files will be stored in `/app/design/frontend/<interface>/<theme>/template/example/`

Any admin template files for the frontend of our module will be stored as follows:

- XML files will be stored in `/app/design/adminhtml/<interface>/<theme>/layout/example/`
- Output files will be stored in `/app/design/adminhtml/<interface>/<theme>/template/example/`

Here's a breakdown of what each folder is for:

- `Block/`—For processing of all display blocks called by the system for the module. These are controllers that will be called in the XML layout files within a theme, in order to display something.
- `controllers/`—Our controllers that support the application and structurally keep things together.
- `etc/`—Configuration files for the module, for declaring things such as the default options when installed and declaring all blocks, models, and install/upgrade actions.
- `Model/`—For placement of all models to support controllers in the module.
- `sql/`—SQL actions when the module is installed/upgraded/uninstalled.

Zend Framework and its role within Magento

Magento (at its raw PHP base) is built on the Zend Framework. From the database class to the handling of URLs, Magento is in its raw form, with Zend Framework doing all the work. Alongside this, Varien has built several core modules on top of the Zend Framework, in order to tie it altogether into the system as we know it.

What is Zend Framework

Zend Framework's official site best describes the framework as follows:

> *Zend Framework (ZF) is an open source framework for developing web applications and services with PHP 5. ZF is implemented using 100% object-oriented code. The component structure of ZF is somewhat unique; each component is designed with few dependencies on other components. This loosely coupled architecture allows developers to use components individually. We often call this a "use-at-will" design.*

While they can be used separately, Zend Framework components in the standard library form a powerful and extensible web application framework when combined. ZF offers a robust, high performance MVC implementation, a database abstraction that is simple to use, and a forms component that implements HTML form rendering, validation, and filtering so that developers can consolidate all of these operations using one easy-to-use, object-oriented interface. Other components, such as Zend_Auth and Zend_Acl, provide user authentication and authorization against all common credential stores. Still others implement client libraries to simply access to the most popular web services available. Whatever your application needs are, you're likely to find a Zend Framework component that can be used to dramatically reduce development time with a thoroughly tested foundation.

How Zend Framework works

The Zend Framework (at its core) is designed to be used as a package or separate modules. This (among other features) makes it unique, as most other frameworks are designed to be used plainly as frameworks or not at all.

However, the Zend Framework comes with classes that allow us to use it as a standalone framework and develop with it as one. Instead of being delivered with a preset amount of directories and layout for developers, it only suggests a layout for our files. This means that we can adapt the framework to meet our current workflow and choose how much we adapt the workflow to fit the framework.

It's role and effect in Magento

The Zend Framework allows Magento to focus on the core issues at hand. It removes a lot of the work on the database and core structural classes and puts the work towards fixing and adding to core modules of Magento.

Most importantly it gives developers a standard approach to development that they can move across and apply to Magento. The standard development practices help greatly in adopting Magento as a platform and make it easier for developers having experience with Zend Framework to adapt to Magento.

More information on learning the Zend Framework and resources can be found at the back of this book in the Appendix attached. Its official site is located at: `http://framework.zend.com/`.

Backing up Magento's data

It's important to know how to back up our site, to ensure that our installation's data is not lost (if ever things go bad).

It is recommended to back up our Magento installation:

- Regularly as a base to ensure that there are incremental backups of our system
- Before installing new modules or themes from Magento Connect
- When developing modules
- Before upgrading our system

Backing up the files

We will need to back up all the files relating to the Magento installation, when backing up our system. Two of the ways in which this can be done are given below.

Manually

Manually, we are able to download all the files of the installation to our hard drive. This is the longest method of backing up the files and is the most foolproof method available.

Using SSH

Using SSH, we're able to vastly speed up the duration of backing up the servers. We can do this in two ways:

- Zipping up all files, if the server has it enabled
- Copying all files to another directory

Both of these depend on whether or not our server has SSH. So if this isn't available to us, then we cannot use these methods.

[Both of these methods require us to connect to our server via SSH first and then use the `cd` command to get to the directory (which Magento is installed in), before running the commands.]

Zipping up all files

This will create a `zip` file of our entire Magento installation's files and folders called `magento_archive.zip`.

```
tar cf magento_archive.tar *
```

To `untar` this archive, extract the files afterwards:

```
tar -xvf yourfilename.tar
```

We can then move this to another directory of our choice using the `mv` command:

```
mv magento_archive.zip /path/to/new/destination/
```

Copying all files to another directory

We run the following command to copy all files (as they are) into another directory on our server. We'll replace the full path with the path to the desired directory, into which we want to copy all the files.

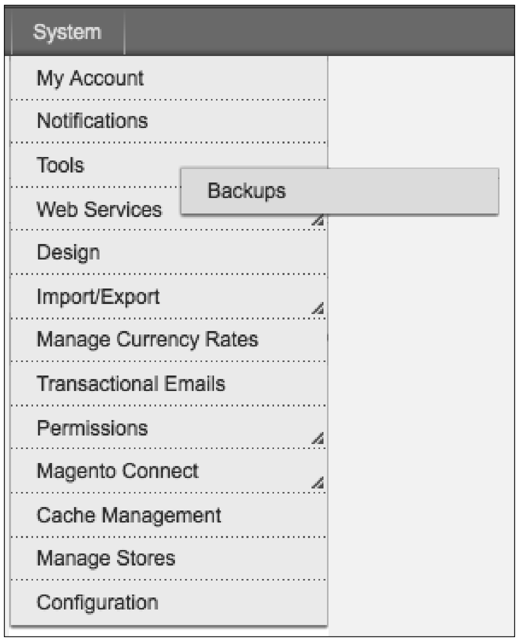
```
cp -R * /path/to/new/destination/
```

Backing up the database

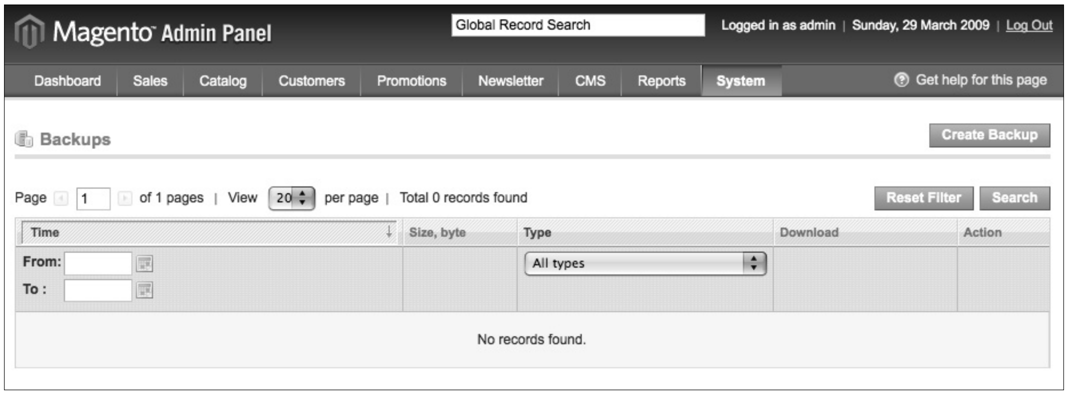
We'll need to back up the database as part of our Magento backup. Let's go through how.

Using the system itself

Magento comes with a built-in method for backing up our installation and keeping several backups logged, in case we want to download older backups at any time. It can be found in the **System** menu under **Tools**:



The initial screen will be similar to the next screenshot:

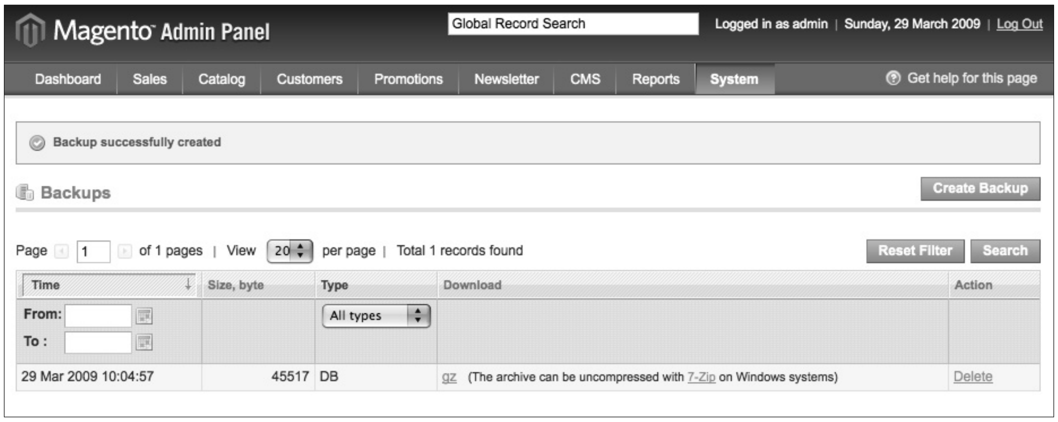


Page 1 of 1 pages | View 20 per page | Total 0 records found

Time	Size, byte	Type	Download	Action
From: To :		All types		

No records found.

To create a backup via the **System** panel, all we need to do is click on **Create Backup** in the upper-right of the screen and wait for it to finish:



Backup successfully created

Page 1 of 1 pages | View 20 per page | Total 1 records found

Time	Size, byte	Type	Download	Action
From: To:		All types		
29 Mar 2009 10:04:57	45517	DB	gz (The archive can be uncompressed with 7-Zip on Windows systems)	Delete

The process of creating a backup can take quite a while (especially for bigger databases), so we will need to keep an eye on our server's memory limits and PHP execution limits. These are set in our `.htaccess` file on runtime, but some servers will only run the defaults and not allow them to be overridden. If we encounter a white screen instead of the success message (shown in the previous screen), then the problem is either memory limit or execution time limit. We will need to increase them ourselves or contact our web host.

Once the backup is completed, however, we'll be able to find it in our `/var/backups/` folder. They will be named by timestamp and the highest numbered filename will be the last to be backed up.

Using phpMyAdmin

The most common back up solution is phpMyAdmin, and some people prefer it over any built-in method. To export via phpMyAdmin, we:

1. Navigate to the database
2. Switch to the **export** tab
3. Select all tables and **SQL** as the export type
4. Under **options** on the right-hand side select **Disable foreign key checks**
5. Select **save as file** at the very bottom of the page
6. If we want to match the compression type of Magento's output, select **gzipped** as our compression method
7. Click the **Go** button to export

This will give us an SQL file, which we can then import at a later date back into an empty database and restore our data.

Summary

In this chapter, we've learned the following:

- How Magento's folder structure and files are laid out
- What each of the base directory's folders and files do
- How the template system works
- How modules work within Magento
- About the Zend Framework and how it benefits Magento
- How best to go about backing up Magento and when to go about it

Further to this chapter, I want you to read the Magento designer's guide and the Zend Framework documentation and examples. There are also a very good group of links for you to read through in the Appendix at the back of this book. These will increase your knowledge of the Magento architecture and benefit you throughout this book.