Documentation for Magento Developers

A Getting Started Guide to Developing Magento Extensions.

Mark Sanborn, Mark Lehm, Ryan Leising, and Richard Fowler

Documentation for Magento Developers

Disclaimer

Written by Published by

- Mark Sanborn
- Mark Lehm
- Ryan Leising
- Richard Fowler

660 York St San Francisco CA 94110

Content

Chapter 1

Introduction

As you probably know Magento is one of the best open source ecommerce platforms out there. It is packed full of great features and allows for almost unlimited flexibility and customization; however, the Magento code base is vast and there is very little current documentation on developing custom extensions. Prior to this book many developers come into the Magento world and immediately find themselves puzzled by a shortage of documentation. The process that many developers take at the beginning usually begins by Google search which almost always turns up a stale blog post on how to implement a specific customization to Magento. Upon further searching developers quickly realize that there is a lack of good guides on how to get started creating a new extension. This is where we come in.

We want to take away that two week pain period that Magento developers often face and replace it with a guide that will introduce you to everything you need to build your extension paired with a quick "Hello World" extension walk-through that will enable you to build your first extension in a matter of minutes. See: *Chapter 3: Your First Extension*

About the Authors

Mark Sanborn

Mark is able to use his developer skills and business knowledge to see pain points and develop simple solutions to complex problems. Mark Sanborn has been writing developer tutorials on his blog to an audiance of a million viewers for over five years. Mark also developed the product, RocketShipIt, that makes integrating custom shipping rates, tracking, and labels on the web easy.

Mark Lhem

Web/UI designer, in-house developer and systems administrator, Mark Lehm is a web design professional offering broad and diverse technical and project leadership talents in full life cycle design and development of corporate websites and web marketing tools.

Richard Fowler

Richard has extensive knowledge in content, SEO/SEM and UX. He is a content strategy and user experience design veteran with expertise in integrating brand strategy, information architecture, SEO and SEM to advance clients' business goals.

What This Book is Not

This book is not intended to be an ultimate reference to every object, class, function, and feature of Magento. Rather it is a no nonsense concise guide to get you developing Magento extensions as fast as possible.

We believe you will be better off knowing how to access this information through code than to look it up in a reference guide that may become stale or obsolete when Magento releases a new version.

Assumptions

We are going to assume you are familiar with the command line, *PHP*, *XML*, *Apache* and setting up a basic Magento installation.

Feedback

If any of the examples in this book are unclear, you find a typo, or just wish the book contained a chapter on subject X please don't hesitate to email us at: **support@vonnda.com**.

Chapter 2

Prepping Your Magento Enviornment for Development

What we hope to accomplish in this section is to turn your Magento installation into an effeciant development enviornment. This enviornement will make your life easier by **turning off caching** so you can see changes immediately, **turning on error reporting** so you can see what went wrong, **turning on logging** so you can capture valuable debug information, and other helpful things like **extending the admin session** so you don't have to constantly log in and a whole lot more.

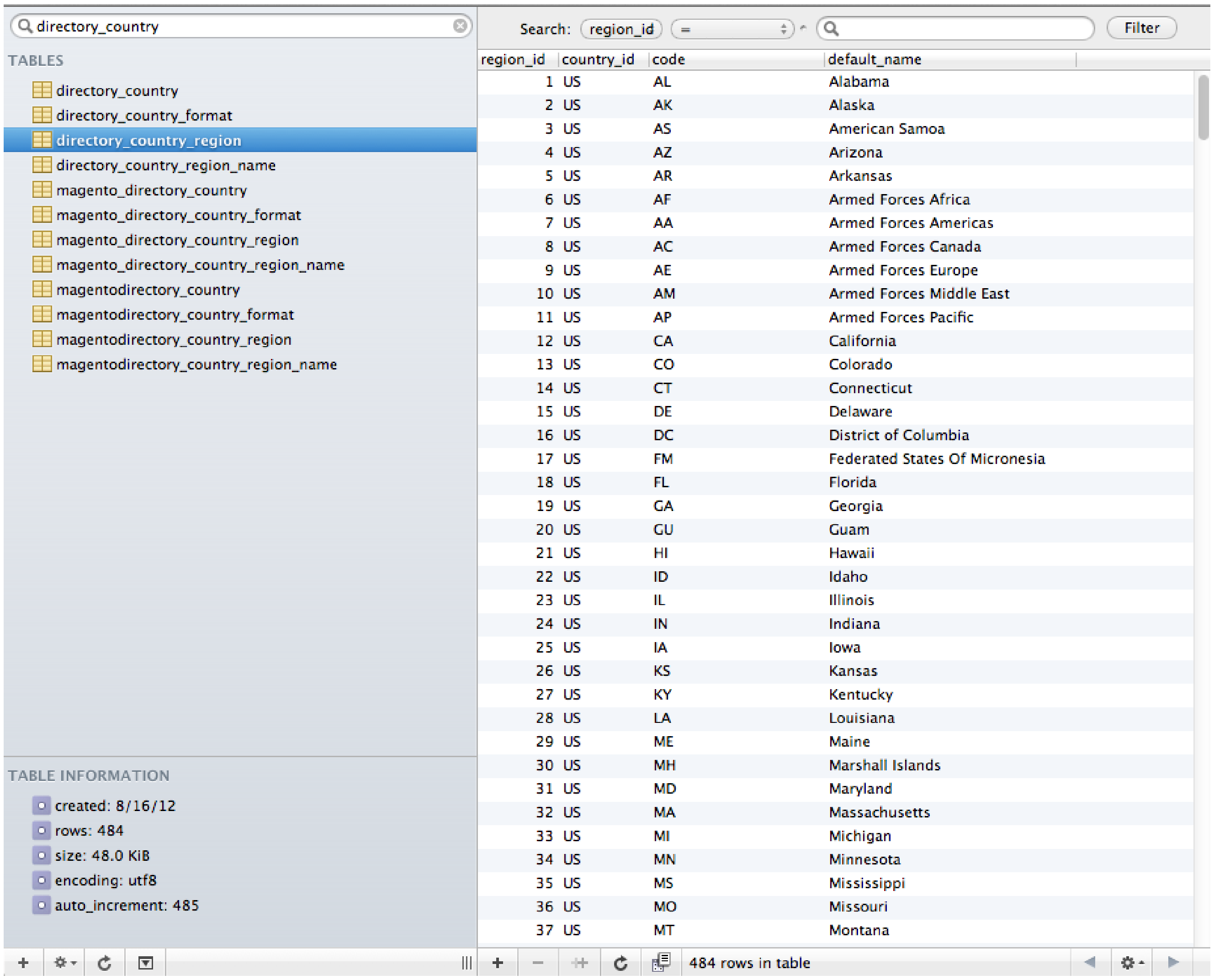
Installing Magento

We are going to assume you have the latest version of Magento installed.

Start by downloading and installing the latest release of Magento from magentocommerce.com.

MySQL Tools

Apart from the actual MySQL server, it is useful to have a GUI client to inspect Magento's database tables from time to time. The tool we most often use around the office is Sequel Pro www.sequelpro.com. You can also use *phpMyAdmin* over Sequel Pro if you are used to that.



directory_country
directory_country_format
directory_country_region
directory_country_region_name
magento_directory_country
magento_directory_country_format
magento_directory_country_region
magento_directory_country_region_name
magentodirectory_country
magentodirectory_country_format
magentodirectory_country_region
magentodirectory_country_region_name

created: 8/16/12
rows: 484
size: 48.0 KiB
encoding: utf8
auto_increment: 485

region_id	country_id	code	default_name
1	US	AL	Alabama
2	US	AK	Alaska
3	US	AS	American Samoa
4	US	AZ	Arizona
5	US	AR	Arkansas
6	US	AF	Armed Forces Africa
7	US	AA	Armed Forces Americas
8	US	AC	Armed Forces Canada
9	US	AE	Armed Forces Europe
10	US	AM	Armed Forces Middle East
11	US	AP	Armed Forces Pacific
12	US	CA	California
13	US	CO	Colorado
14	US	CT	Connecticut
15	US	DE	Delaware
16	US	DC	District of Columbia
17	US	FM	Federated States Of Micronesia
18	US	FL	Florida
19	US	GA	Georgia
20	US	GU	Guam
21	US	HI	Hawaii
22	US	ID	Idaho
23	US	IL	Illinois
24	US	IN	Indiana
25	US	IA	Iowa
26	US	KS	Kansas
27	US	KY	Kentucky
28	US	LA	Louisiana
29	US	ME	Maine
30	US	MH	Marshall Islands
31	US	MD	Maryland
32	US	MA	Massachusetts
33	US	MI	Michigan
34	US	MN	Minnesota
35	US	MS	Mississippi
36	US	MO	Missouri
37	US	MT	Montana

Setting the Time

The first thing we want to do is make sure that our Magento installation and server have the approriate date and time. **If your server time is too far into the future or past it breaks a core functionality in Magento and in some cases prevents you from being able to log in**.

Make sure your date/time is properly set by opening a terminal on the server hosting Magento and type, `date`.

If your date or time is off, this command will get it back in sync if you have **ntpdate** on your system:

```
sudo ntpdate us.pool.ntp.org
```

Turn on Symlinks

Magento expects your extension files to be scattered around in different folders. This not only causes developer frustration it makes keeping your extensions under version control a pain. Later in the book we will show you how to easily manage your extensions with version control via symlinks. For this to work we need to turn on symlinks.

To turn symlinks on for template files:

System > **Configuration** > **Developer** > **Template Settings**

Make sure Allow Symlinks is set to **Yes**

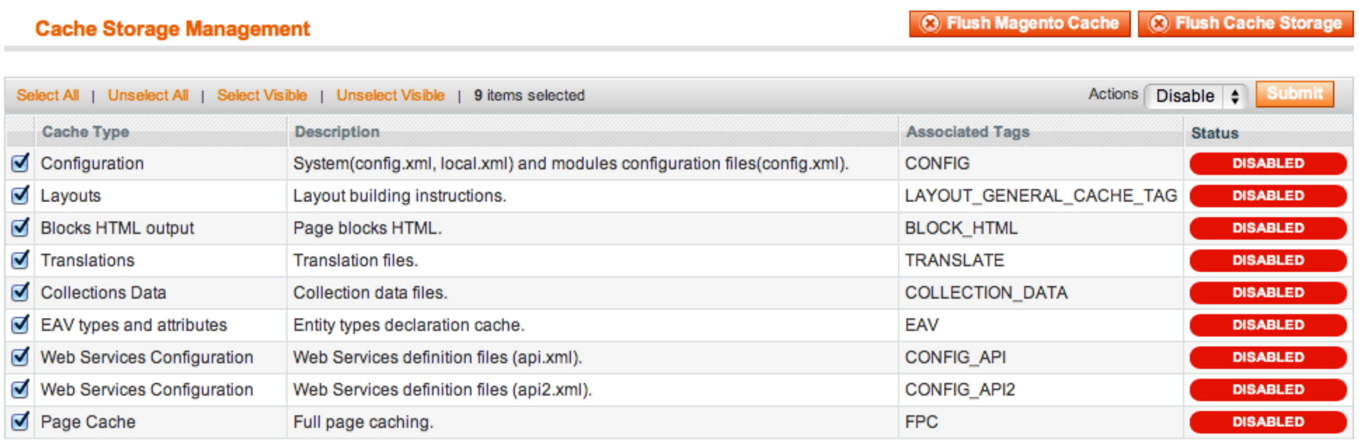
Turn Caching Off

Nothing is more frustrating than making code changes expecting a different result only to find out that caching has been turned on the whole time. When developing your extensions you should **always have caching turned off**.

Login to the backend, go to:

System > **Cache Management** > **Select All** > **Disable** > **Submit**

Click *Flush Magento Cache*



	Cache Type	Description	Associated Tags	Status
✓	Configuration	System(config.xml, local.xml) and modules configuration files(config.xml).	CONFIG	DISABLED
✓	Layouts	Layout building instructions.	LAYOUT_GENERAL_CACHE_TAG	DISABLED
✓	Blocks HTML output	Page blocks HTML.	BLOCK_HTML	DISABLED
✓	Translations	Translation files.	TRANSLATE	DISABLED
✓	Collections Data	Collection data files.	COLLECTION_DATA	DISABLED
✓	EAV types and attributes	Entity types declaration cache.	EAV	DISABLED
✓	Web Services Configuration	Web Services definition files (api.xml).	CONFIG_API	DISABLED
✓	Web Services Configuration	Web Services definition files (api2.xml).	CONFIG_API2	DISABLED
✓	Page Cache	Full page caching.	FPC	DISABLED

Turning Error Reporting On

By default Magento tries to suppress error messages. In a production enviornment this is a good thing; however, for developing we need error reporting enabled.

Open up `index.php` and uncomment:

```
ini_set('display_errors', 1);
```

Then set Magento to *Developer Mode* by changing:

```
if (isset($_SERVER['MAGE_IS_DEVELOPER_MODE'])) {
    Mage::setIsDeveloperMode(true);
}
```

to this:

```
//if (isset($_SERVER['MAGE_IS_DEVELOPER_MODE'])) {
    Mage::setIsDeveloperMode(true);
//}
```

Note: You could also set the `$_SERVER` variable but we found this method unreliable.

Turn Logging On

We also need to have logging turned on.

System > **Configuration** > **Developer**

Expand the **Log Settings**.

Change to **Enabled** and leave the names default.

Extend the Session Timeout

Magento has a default session timeout of **15 minutes** which means you will be logged out of the admin backend every 15 minutes which is quite annoying during development. Extend the timeout by going to:

System > **Configuration**

System > **Advanced** > **Admin**

Expand **Security**

Default is 900 seconds which is 15 minutes. Set Session Lifetime (seconds) to 86400.

This will log you out of your current session but next login will stay connected for 24 hours.

Depending on your server configuration you may also need to set your maxlifetime session expiration time in your `php.ini` file for this to have an effect.

```
php_value session.gc_maxlifetime 86400
```

Create a Restore Point with Magentomatic

In this section we will breifly introduce Magentomatic and make a backup of your Magento installation so you have a clean database of a fresh Magento installation if you ever need to roll back for any reason.

Magentomatic is a tool developed by Vonnda to automatically manage common developer tasks in Magento like:

- adding an admin user
- backing up the database
- restoring the database
- deleting users/products/orders
- setting config values
- clearing the cache
- verifying integrity of core files
- and more

Make a Database Backup

Make sure you have Magentomatic extracted into your Magento installation.

Change directories to `magentomatic`

```
php magentomatic.php backup
```

and follow the prompts to make a backup of the database.

For a list of commands type `php magentomatic.php`

Chapter 3

Your First Extension

Now that you've gotten a start on basic setup and configuration of your magento installation, we're going to throw you right into the thick of it and guide you through building your first extension. Don't worry, there will be explanations along the way, and later chapters delve deeper into the inner workings.

Hello World!

We are first going to initialize our extension manually so you can see how the process works. Later we will show you how we can use a tool to create these folders and files automatically.

Step 1: Create a "Mycompany" folder under app/code/local

```
app
└── code
    └── local
        └── Mycompany
```

Step 2: Create a "Myfirstmodule" folder.

```
app
└── code
    └── local
        └── Mycompany
            └── Myfirstmodule
```

Step 3: Create the following folders inside (case matters!): Block, controllers, etc, Helper, Model, sql.

```
app
└── code
    └── local
        └── Mycompany
            └── Myfirstmodule
                ├── Block
                ├── controllers
                ├── etc
                ├── Helper
                ├── Model
                └── sql
```

Step 4: Under the etc folder create the file, `config.xml`.

```
app
└── code
    └── local
        └── Mycompany
            └── Myfirstmodule
                ├── Block
                ├── controllers
                ├── etc
                |    └── config.xml
                ├── Helper
                ├── Model
                └── sql
```

Step 5: Fill `config.xml` with:

```
<?xml version="1.0"?>
<config>
  <modules>
    <Mycompany_Myfirstmodule>
      <version>0.1.0</version>
    </Mycompany_Myfirstmodule>
  </modules>
</config>
```

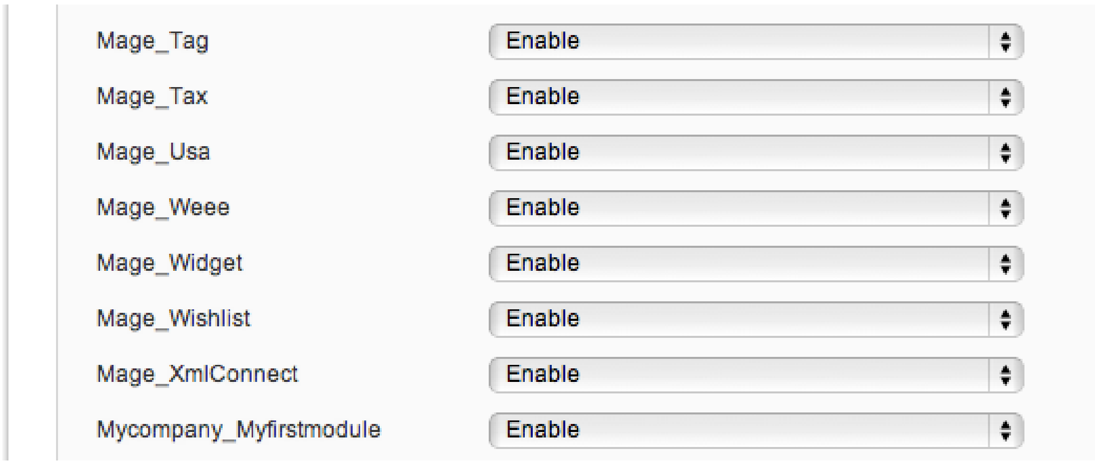
Step 6: Under, etc/modules create the file, `Mycompany_Myfirstmodule.xml`

```
app
├── code
|    └── local
|        └── Mycompany
|            └── Myfirstmodule
|                ├── Block
|                ├── controllers
|                ├── etc
|                |    └── config.xml
|                ├── Helper
|                ├── Model
|                └── sql
└── etc
    └── modules
        └── Mycompany_Myfirstmodule.xml
```

Step 7: Fill `Mycompany_Myfirstmodule.xml` with:

```
<?xml version="1.0"?>
<config>
  <modules>
      <Mycompany_Myfirstmodule>
        <!-- This activates the module, flip to false to deactivate -->
        <active>true</active>
        <codePool>local</codePool><!-- Remember we talked about code pools -->
      </Mycompany_Myfirstmodule>
  </modules>
</config>
```

Step 8: Make sure Magento is loading your extension, System > Configuration > Advanced



Mage_Tag	Enable
Mage_Tax	Enable
Mage_Usa	Enable
Mage_Weee	Enable
Mage_Widget	Enable
Mage_Wishlist	Enable
Mage_XmlConnect	Enable
Mycompany_Myfirstmodule	Enable

If you see the name of your module listed it means you have followed the above steps correctly and Magento has loaded it.

Congratulations your module is installed!!

Write Your First Controller

Step 1: Modify your `config.xml` file in your module's etc folder to include a new frontend router:

```
<?xml version="1.0"?>
<config>
    <modules>
        <Mycompany_Myfirstmodule>
```

```
            <version>0.1.0</version>
        </Mycompany_Myfirstmodule>
    </modules>
    <frontend>
        <routers>
            <myfirstmodule>
                <use>standard</use>
                <args>
                    <module>Mycompany_Myfirstmodule</module>
                    <frontName>myfirstmodule</frontName>
                </args>
            </myfirstmodule>
        </routers>
    </frontend>
</config>
```

Step 2: Create a new file, `IndexController.php` under your controllers folder and fill it with this:

```
class Mycompany_Myfirstmodule_IndexController extends Mage_Core_Controller_Front_Action {
    public function indexAction() {
        echo 'Hello World!';
    }
}
```

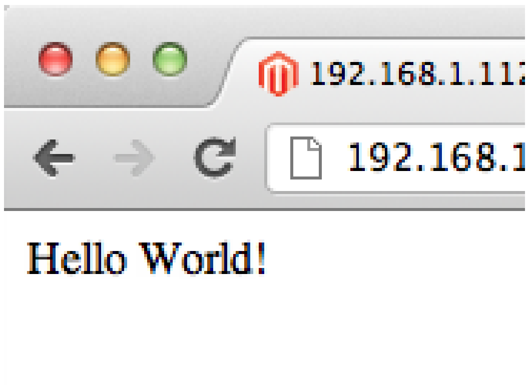
Note: Our function has the name indexAction, the first part 'index' will be used as part of the url.

Step 3: Go to http://yourmageinstallation/myfirstmodule/index/index/

```
Go to http://yourmageinstallation/myfirstmodule/index/index/
                                         ^        ^     ^
                                         |        |     |
                                         |        |     Function name minus 'Action'
                                         |        |
                                         |        Controller name minus 'Controller.php'
                                         |
                                         frontName in config.xml
```

Note: Since this is our index controller you could also reach this url here:

http://yourmageinstallation/myfirstmodule/



Hello World!

GET and POST Variables

You can gain access to both *get* and *post* variables with:

```
$params = $this->getRequest()->getParams();
```

So if we were to change our "Hello World!" message in the `IndexControler.php` file to:

```
public function indexAction() {
    print_r($this->getRequest()->getParams());
    echo 'Hello World!';
}
```

and go to the following url: **http://yourmageinstallation/myfirstmodule/?myvariable=22&another=something** , we would see:

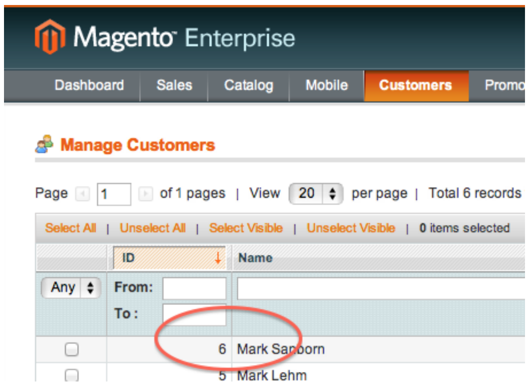
```
Array ( [myvariable] => 22 [another] => something ) Hello World!
```

Notice what happens when you go to this url: **http://yourmageinstallation/myfirstmodule/index/index/id/22**

```
Array ( [id] => 22 ) Hello World!
```

This means we can also pass *get* variables to Magento with clean seo friendly urls!

Go ahead and create a customer on the frontend and look up your newly created customer's ID.



Page 1 of 1 pages | View 20 per page | Total 6 records

Select All | Unselect All | Select Visible | Unselect Visible | 0 items selected

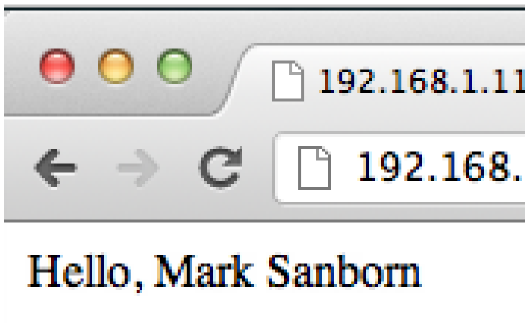
	ID	Name
Any	From: To:	
☐	6	Mark Sanborn
☐	5	Mark Lehm

Lets now modify our controller slightly:

```
public function indexAction() {
    $params = $this->getRequest()->getParams();
    $customer = Mage::getModel('customer/customer')->load($params['id']);
    echo 'Hello, '. $customer->getName();
}
```

Now go to the url substituing the id with your newly created customer's id:

Mine: **http://yourmageinstallation/myfirstmodule/index/index/id/6**



Hello, Mark Sanborn

Congratulate yourself, you now have a working url that accepts *get* parameters and loads your first Magento customer object. When developing your extensions you will primarily deal with *customer*, *order*, and *product* objects.

Writing Output to the Log

So you have loaded your first Magento customer object but it is not fair, we told you what to put in your controller. How were you supposed to know that `$customer->getName();` would produce the customer's name? What else can we do with customer objects?

We would like to take this opportunity and introduce the Magento log and show you how you can determine what data you have access to.

Make sure your logs are turned on from **Prepping Your Magento Enviornment for Development** chapter.

Modify your controller and add: `Mage::log($customer);`

```
public function indexAction() {
    $params = $this->getRequest()->getParams();
    $customer = Mage::getModel('customer/customer')->load($params['id']);
    Mage::log($customer->getData());
    echo 'Hello, '. $customer->getName();
}
```

Open up `var/log/system.log` and scroll to the bottom:

When developing you will probably want to have this refresh automatically when new output is appended to the log. On Unix like systems you can use `tail`

```
tail -f var/log/system.log
```

Explore Magento Objects/Snippits

Take this time now to play around with some of the snippits in the *Recipes* chapter and try to output various magento objects from the controller or output them to the log.

Example

```
public function indexAction() {
    $customer = Mage::getModel('customer/customer')->load($params['id']);
    print_r($customer->getData()); // outputs customer data
    Mage::log(get_class_methods($customer)); // writes available customer methods to log
}
```

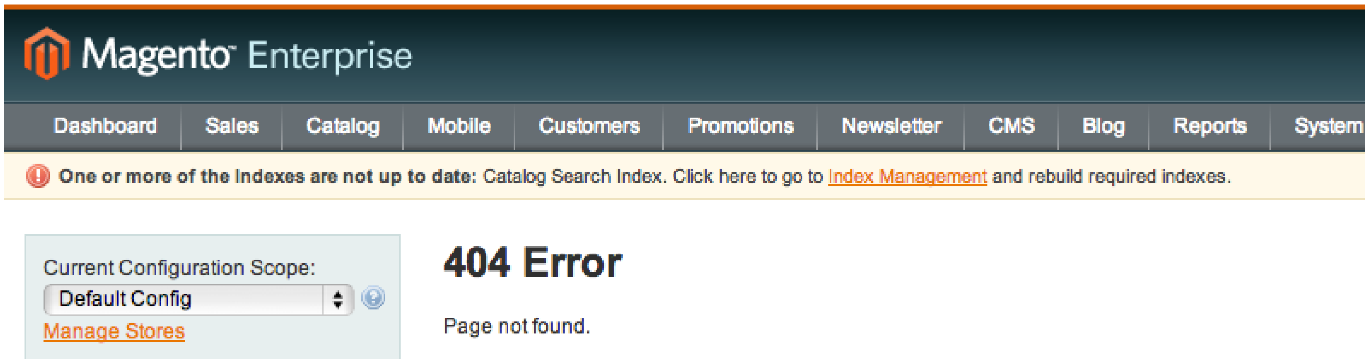
A Simple Admin Interface

We are now going to add a super simple admin interface to our extension. In later chapters we will explore the admin area in more detail.

Create the file `system.xml` under the `etc` folder:

```
<?xml version="1.0"?>
<config>
  <tabs>
      <mycompany_extensions translate="label">
      <label>My Companies Extensions</label>
      <sort_order>210</sort_order>
      </mycompany_extensions>
  </tabs>
  <sections>
    <mymodule translate="label">
      <label>My Module</label>
      <tab>mycompany_extensions</tab>
      <frontend_type>text</frontend_type>
      <sort_order>1000</sort_order>
      <show_in_default>1</show_in_default>
      <show_in_website>1</show_in_website>
      <show_in_store>1</show_in_store>
    </mymodule>
  </sections>
</config>
```

At this point we have a menu section and tab for our extension in the backend; however, if we were to try to view it we would see a 404 error. This is Magento's way of saying that we don't have permission to view this section. It is one of the common idiosyncrasies of working with Magento extensions.



One or more of the Indexes are not up to date: Catalog Search Index. Click here to go to Index Management and rebuild required indexes.

404 Error

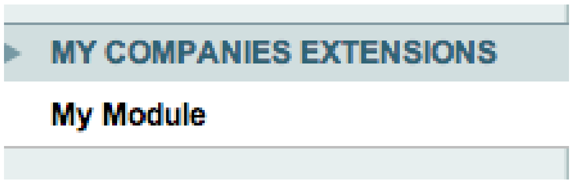
Page not found.

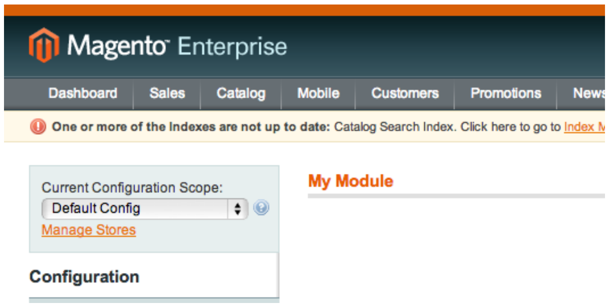
To add the appropriate permissions add the file `adminhtml.xml` under the `etc` folder.

```
<?xml version="1.0"?>
<config>
  <acl>
    <resources>
      <all>
        <title>Allow Everything</title>
      </all>
      <admin>
        <children>
          <system>
```

```
            <children>
              <config>
                <children>
                  <mymodule translate="title">
                    <title>My Module</title>
                    <sort_order>100</sort_order>
                  </mymodule>
                </children>
              </config>
            </children>
          </system>
        </children>
      </admin>
    </resources>
  </acl>
</config>
```

Now logout to make sure Magento refreshes permissions and you should see your module and admin section.





One or more of the Indexes are not up to date: Catalog Search Index. Click here to go to Index M

Current Configuration Scope: Default Config

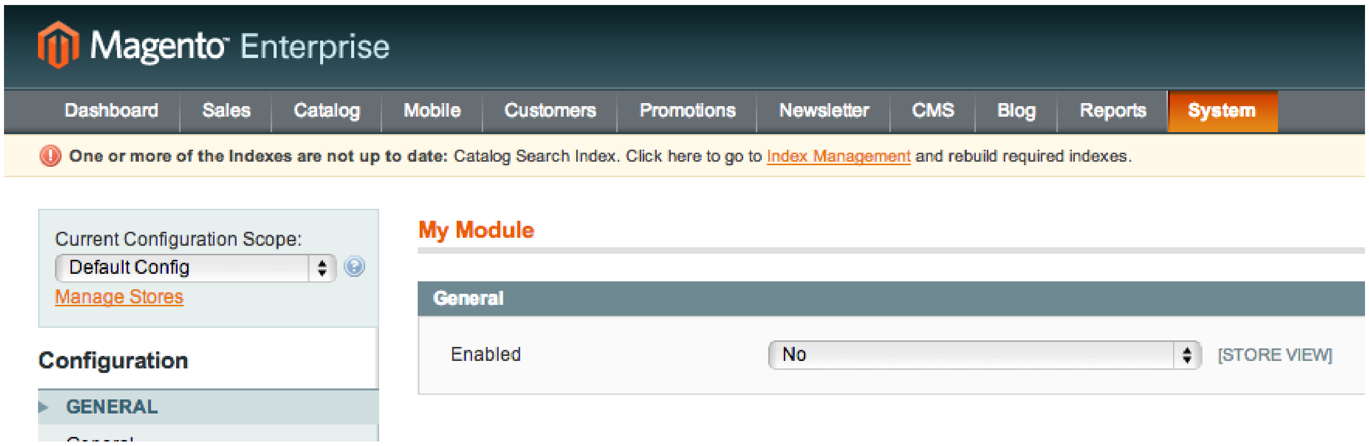
We are now ready to add configuration settings for our extension.

Add in the following xml into `system.xml` under `<config><sections><mymodule>`

```
<groups>
    <!-- Group of config options -->
    <general translate="label">
        <!-- Title of group -->
        <label>General</label>
        <frontend_type>text</frontend_type>
        <!-- Sort order -->
        <sort_order>10</sort_order>
        <!-- Visibility options -->
        <show_in_default>1</show_in_default>
        <show_in_website>1</show_in_website>
        <show_in_store>1</show_in_store>
        <!-- Expand group area by default -->
        <expanded/>

        <!-- Fields go here -->
        <fields>
```

```
                <enabled translate="label">
                    <!-- Label for field -->
                    <label>Enabled</label>
                    <!-- Type of field -->
                    <frontend_type>select</frontend_type>
                    <source_model>adminhtml/system_config_source_yesno</source_model>
                    <sort_order>1</sort_order>
                    <show_in_default>1</show_in_default>
                    <show_in_website>1</show_in_website>
                    <show_in_store>1</show_in_store>
                </enabled>
            </fields>

        </general>
    </groups>
```

At this point you should have a *General* group with a single *field* called `enabled`



One or more of the Indexes are not up to date: Catalog Search Index. Click here to go to Index Management and rebuild required indexes.

Current Configuration Scope: Default Config
Manage Stores

Configuration
GENERAL

My Module
General
Enabled: No [STORE VIEW]

You can access the value of this field using `getStoreConfig()`. The path will follow your xml under the `<sections>` element.

```
    Mage::getStoreConfig('myfirstmodule/general/enabled');
```

For fields that are boolean Magento also gives us the function `getStoreConfigFlag()`

Lets modify our controller to include our new field:

```
    class Mycompany_Myfirstmodule_IndexController extends Mage_Core_Controller_Front_Action {
        public function indexAction() {
            // Check to see if module is enabled!
            if (Mage::getStoreConfigFlag('myfirstmodule/general/enabled')) {
                $params = $this->getRequest()->getParams();
                $customer = Mage::getModel('customer/customer')->load($params['id']);
                Mage::log($customer->getData());
                echo 'Hello, '. $customer->getName();
            } else {
                echo 'This module is disabled.';
            }
        }
    }
```

Now try toggling the enabled field in the admin backend and see how your controller responds.

You now have a small taste for all of the essential bits that make up a full Magento module. These next chapters will be devoted to diving into each section in more detail.

Chapter 4

Understanding the Magento Directory Structure

Code Pools

Magento Extensions are located in one of three code pools. The code pools are: **core**, **local**, and **community**. All of the modules distributed with the base Magento are in the core code pool. All of the custom modules that you develop can be installed in the local code pool. Although you can install any module in local as well as community it is best to put third-party modules under community and change their xml file accordingly to keep third party modules from overriding the files you are trying to override in local.

Magento will first look in **local**, then **community**, then **core**.

```
app/
└── code
    ├── community
    ├── core
    └── local
```

Sample app directory folder

```
app/
├── code
|   └── local
|       └── Yourcompany
|           └── Yourmodule
|               ├── Block
|               ├── controllers
|               |   └── IndexController.php
|               ├── etc
|               |   └── config.xml
|               ├── Helper
|               ├── Model
|               └── sql
└── etc
    └── modules
        └── Yourcompany_Yourmodule.xml
```

Generate These Files Automatically

Verify Your Extension is Running

Go to **System** > **Configuration** > **Advanced**

Click the **Disable Modules Output** header.

Find your extension in the list.

Congratulations your module is installed and active!

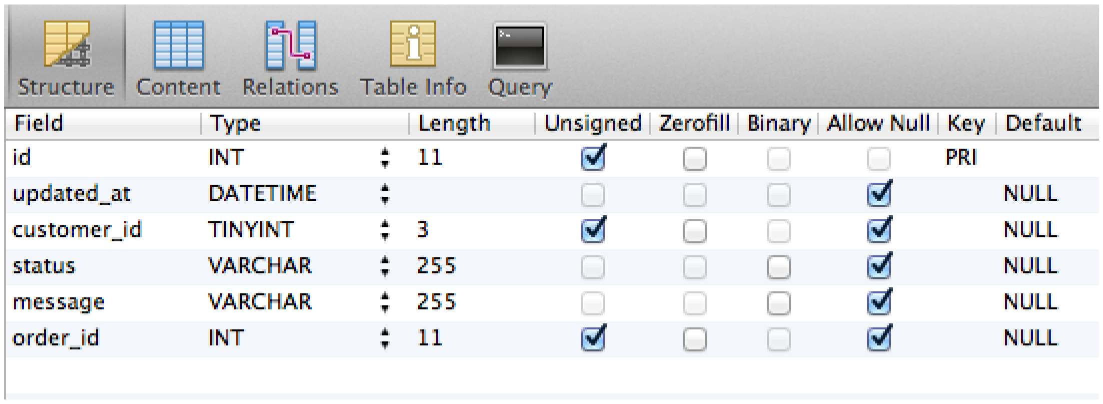
Chapter 5

Models

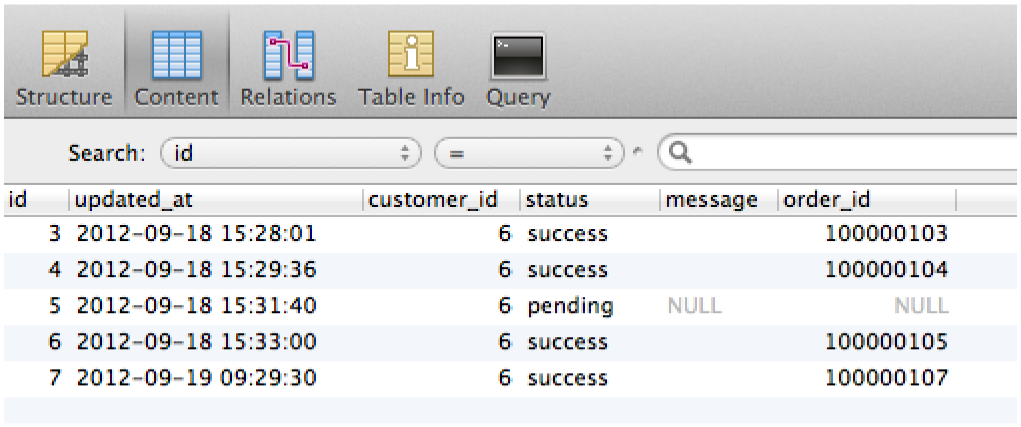
It is the Model's job to help move data from the database into the framework. Unlike some MVC approaches Magento's models are mainly responsible for manipulating the data where previously this might be done in the controller. The output, or rendering of the model data is done by the Blocks.

Creating a Model

First start out by creating your table in the database with your favorite tool.



Field	Type	Length	Unsigned	Zerofill	Binary	Allow Null	Key	Default
id	INT	11	✓				PRI	
updated_at	DATETIME					✓		NULL
customer_id	TINYINT	3	✓			✓		NULL
status	VARCHAR	255				✓		NULL
message	VARCHAR	255				✓		NULL
order_id	INT	11	✓			✓		NULL



id	updated_at	customer_id	status	message	order_id
3	2012-09-18 15:28:01	6	success		100000103
4	2012-09-18 15:29:36	6	success		100000104
5	2012-09-18 15:31:40	6	pending	NULL	NULL
6	2012-09-18 15:33:00	6	success		100000105
7	2012-09-19 09:29:30	6	success		100000107

Under your *Model* folder create the file `Yourmodule.php`.

```
class Yourcompany_Yourmodule_Model_Yourmodule extends Mage_Core_Model_Abstract{
    public function _construct(){
        parent::_construct();
        $this->_init('yourmodule/yourmodule');
    }
}
```

Then Create the folder *Mysql4* and create another file `Yourmodule.php` where `id` is your primary key.

```
class Yourcompany_Yourmodule_Model_Mysql4_Yourmodule extends Mage_Core_Model_Mysql4_Abstract{
        public function _construct(){
                $this->_init('yourmodule/yourmodule', 'id');
        }
}
```

Create the folder *Yourmodule* under the *Mysql4* folder you just created and add the file `Collection.php`.

```
class Yourcompany_Yourmodule_Model_Mysql4_Yourmodule_Collection extends
Mage_Core_Model_Mysql4_Collection_Abstract{
     public function _construct(){
         $this->_init('yourmodule/yourmodule');
     }
}
```

The end result should look like this:

```
Model/
├── Yourmodule.php
└── Mysql4
    ├── Yourmodule
    |   └── Collection.php
    └── Yourmodule.php
```

Finally add the following into your module's `config.xml`

```
<?xml version="1.0"?>
<config>
    ...
    <global>
        ...
        <models>
            <yourmodule>
                <class>Yourcompany_Yourmodule_Model</class>
                <resourceModel>yourmodule_mysql4</resourceModel>
            </yourmodule>
            <yourmodule_mysql4>
                <class>Yourcompany_Yourmodule_Model_Mysql4</class>
                <entities>
                    <yourmodule>
                        <table>your_table_name</table>
                    </yourmodule>
                </entities>
            </yourmodule_mysql4>
        </models>
        <resources>
            <yourmodule_write>
                <connection>
                    <use>core_write</use>
                </connection>
            </yourmodule_write>
            <yourmodule_read>
                <connection>
                    <use>core_read</use>
                </connection>
            </yourmodule_read>
        </resources>
        ...
    </global>
    ...
</config>
```

Verify that it works by adding this to a controller:

```
$myModel = Mage::getModel('yourmodule/yourmodule');
print_r($myModel);
```

In the event that your extension grows, or you are building a new extension that requires creating more than one model, our current model naming convention of `yourmodule/yourmodule` becomes a little unclear.

```
<models>
    <yourmodule> <!-- model top node -->
        <class>Yourcompany_Yourmodule_Model</class>
        <resourceModel>yourmodule_mysql4</resourceModel>
    </yourmodule>
    <yourmodule_mysql4>
        <class>Yourcompany_Yourmodule_Model_Mysql4</class>
        <entities>
            <tablenode1> <!-- table top node -->
                <table>your_table_name</table>
            </tablenode1>
            <tablenode2> <!-- table top node -->
                <table>your_other_table_name</table>
            </tablenode2>
        </entities>
    </yourmodule_mysql4>
</models>
```

Here we have changed the node containing the table name. Now each table will have its own model and will be accessed differently, `yourmodule/tablenode1` and `yourmodule/tablenode2`. The name of your model top node node will always be the first half of the call to your model and depending on which table you want the second half of the call to your model will be a table top node. The files created earlier will need to be duplicated for each table.

Collections and Querying

`addAttributeToFilter()` is a function that can be called on a product collection in Magento. In short, it adds a condition to the WHERE part of the MySQL query used to extract a collection from the database.

```
$_products = Mage::getModel('catalog/product')->getCollection()
   ->addAttributeToSelect(array('name', 'product_url', 'small_image'))
   ->addAttributeToFilter('sku', array('like' => 'UX%'))
    ->load();
```

The above code would get a product collection, with each product having it's name, url, price and small image loaded in it's data array. The product collection would be filtered and contain only products that have an SKU starting with UX.

Getting First and Last Items

```
$products = Mage::getModel('catalog/product')->getCollection()
var_dump($products->getFirstItem()->getData());
var_dump($products->getLastItem()->getData());
```

Limiting the Column Data

Using `addAttributeToSelect()` we can limit the data.

```
$products = Mage::getModel('catalog/product')
    ->getCollection()
    ->addAttributeToSelect('*'); // would return all
```

Where as

```
$products = Mage::getModel('catalog/product')
    ->getCollection()
    ->addAttributeToSelect('price');
    ->addAttributeToSelect('description');
    // would return price and description data
```

Try these yourself and see the difference with

```
foreach ($products as $product) {
    var_dump($product->getData());
}
```

Filtering

We can filter our collections with the `addFieldToFilter()` function. In SQL you can think of these as your *WHERE* clauses.

Lets say we want to get a product by the sku *SNEW0512*

```
$products = Mage::getModel('catalog/product')->getCollection();
$products->addFieldToFilter('sku','SNEW0512');
$product = $products->getFirstItem();
var_dump($product->getData());
```

Lets try the **Greater Than** conditional. Simply supply an array with the conditional type and value.

Products greater than $100:

```
$products = Mage::getModel('catalog/product')->getCollection();
$products->addFieldToFilter('price', array('gt' => '100'));
var_dump($products>getData());
```

Magento gives us lots of filter conditionals to choose from see: *addAttributeToFilter Conditionals*

Debugging the Query

When developing it is often helpful to see what SQL Magento is using in the backround. You can see what Magento is using to query the database by using the `getSelect()` function.

```
echo (string) $collection->getSelect();
```

For example:

```
$products = Mage::getModel('catalog/product')->getCollection();
echo (string) $products->getSelect();
```

Would output: `` SELECT `e`.* FROM `magento_catalog_product_entity` AS `e` ``

addAttributeToFilter Conditionals

Equals: eq

```
$_products->addAttributeToFilter('status', array('eq' => 1));
```

Not Equals - neq

```
$_products->addAttributeToFilter('sku', array('neq' => 'test-product'));
```

Like - like

```
$_products->addAttributeToFilter('sku', array('like' => 'UX%'));
```

One thing to note about like is that you can include SQL wildcard characters such as the percent sign.

Not Like - nlike

```
$_products->addAttributeToFilter('sku', array('nlike' => 'err-prod%'));
```

In - in

```
$_products->addAttributeToFilter('id', array('in' => array(1,4,74,98)));
```

When using in, the value parameter accepts an array of values.

Not In - nin

```
$_products->addAttributeToFilter('id', array('nin' => array(1,4,74,98)));
```

NULL - null

```
$_products->addAttributeToFilter('description', 'null');
```

Not NULL - notnull

```
$_products->addAttributeToFilter('description', 'notnull');
```

Greater Than - gt

```
$_products->addAttributeToFilter('id', array('gt' => 5));
```

Less Than - lt

```
$_products->addAttributeToFilter('id', array('lt' => 5));
```

Greater Than or Equals To- gteq

```
$_products->addAttributeToFilter('id', array('gteq' => 5));
```

Less Than or Equals To - lteq

```
$_products->addAttributeToFilter('id', array('lteq' => 5));
```

Creating the DB Table

We recommend creating the database table via command line or your favorite SQL tool and then dumping the SQL out and creating the installer script when your extension is complete.

In your extension's directory create a folder in the `sql` folder called: `modulename_setup`.

In `modulename_setup` create the file `mysql4-install-1.0.0.php`.

```
$installer = $this;
$installer->startSetup();
$installer->run("
        YOURSQLGOESHERE
");
$installer->endSetup();
```

Replace YOURSQLGOESHERE with the sql dumped form the command line or your SQL tool.

Create: `Model > Resource > Mysql4 > Setup.php`

```
class Yourcompany_Yourmodule_Model_Resource_Mysql4_Setup extends Mage_Core_Model_Resource_Setup {
}
```

Finally, add the resources node to your modules `config.xml`. This will tell magento what model to use when your module is installed, or if it finds the version of your module has changed.

```
<config>
    ...
    <global>
        ...
        <resources>
            <yourmodule_setup>
                <setup>
                    <module>Yourcompany_Yourmodule</module>
                    <class>Yourcompany_Yourmodule_Model_Resource_Mysql4_Setup</class>
                </setup>
            </yourmodule_setup>
            <connection>
                <use>core_setup</use>
            </connection>
        <resources>
        ...
    </global>
    ...
</config>
```

Overriding a Model

Many times we need to implement new functionality of existing Magento core classes, but we don't want to modify core classes as this might break during Magento ugrades. This is where overriding comes in, also known as "rewrite" or "extend".

Start by finding the model you want to override in the `app/core/Mage` folder.

Copy the directory structure into your module's model dir.

i.e. overrideing the Order model in `app/code/Mage/Sales/Model/Order.php`

`app/code/local/Yourstore/Yourmodule/Model/Sales/Model/Order.php`

Strip out everything inside the class in the file you just copied except the function(s) you want to override. In this example I am looking for `_setState` function because I want to activate code every time an order status changes.

You will be left with something like:

```
class Mage_Sales_Model_Order extends Mage_Sales_Model_Abstract {
}
```

You will be extending Mage_Sales_Model_Order in the same way Magento extended Mage_Sales_Model_Abstract so change this to:

```
class WHAT_DO_I_PUT_HERE extends Mage_Sales_Model_Order {
}
```

WHAT_DO_I_PUT_HERE will be replaced by Yourcompany followed by underscore, Yourmodule, and then the directory structure you have in your `app/code/local/Yourcompany/Yourmodule/` realtive to the `Order.php` file.

Example:

```
Yourcompany_Yourmodule_Model_Sales_Model_Order
```

Make your desired changes to the function you have left in your new class.

Last step: add the override into your config.xml

```
...
<global>
    <models>
        <yourmodule>
            <class>Model</class>
        <yourmodule>
        <sales><!-- Notice the directory structure -->
            <rewrite>
                <!-- order tag is used here as it is under sales -->
                <order>Yourcompany_Yourmodule_Model_Sales_Model_Order</order>
            </rewrite>
        </sales>
    </models>
...
```

Troubleshooting

Not working? Try making a syntax error in your newly created class and see if you get fatal errors.

If you dont get the error you need to make some changes to your `config.xml` file.

Alternative Way to Override a Model

Although it is not as robust and not extension specific you can also override a core model by replicating the folder structure under the `app/code/local` directory. This is a step above modifying the core file itself, which you should never do.

For example if you wanted to override the model `app/code/core/Mage/Customer/Model/Address.php` you could simply copy that file into `app/code/local/Mage/Customer/Model/Address.php`.

You can ensure your file is being called by making an intentional PHP syntax error and going to a page that calls that model. In our example if we make an intentional PHP syntax error, browse to the backend, and open a customer we would see the fatal error. This proves that Magento is calling our modified file and not the one in `app/code/core`.

If you don't want to override simply delete/rename the folder under `app/code/local/Mage` and Magento will continue loading from `app/code/core/Mage`.

Although the traditional way of overriding a model is better this method is great if you simply want to do a quick test. You can always pull the model out of the local folder and include it in the extension later.

Chapter 6
Blocks

As you will see later in the Controller section in general Magento's Controller does not pass a data object to the view or set properties on the view object. Instead, the view will directly reference a system model to get the information necessary for rendering output.

Basically this means that the Magento's view in the traditional MVC pattern has been seperated into *Blocks* and *Templates*

Magento blocks and templates go hand in hand for each block you will have a corresponding .phtml template file.

Overriding a Block

Copy the folder structure of the core block you want to override under your `Block` folder and copy over the file.

For example if I wanted to override the core file: `code/core/Mage/Adminhtml/Block/Catalog/Product/Attribute/Edit/Tab/Main.php`

```
Block/
└── Adminhtml
    └── Block
        └── Catalog
            └── Product
                └── Attribute
                    └── Edit
                        └── Tab
                            └── Main.php
```

Open up your copy of `Main.php` and modify the class name to match the structure and extend the original class:

```
class Yourcompany_Yourmodule_Block_Adminhtml_Block_Catalog_Product_Attribute_Edit_Tab_Main extends
Mage_Adminhtml_Block_Catalog_Product_Attribute_Edit_Tab_Main
```

and finally add to your `config.php`

```
<?xml version="1.0"?>
<config>
    ...
    <global>
        <blocks>
            <adminhtml>
                <rewrite>

<catalog_product_attribute_edit_tab_main>Yourcompany_Yourmodule_Block_Adminhtml_Block_Catalog_Product_Attribute_Edit_Tab_
                </rewrite>
            </adminhtml>
        </blocks>
    </global>
...
</config>
```

Chapter 7

Controllers

In general the controller's job is to alter Models, and then tell the system it's layout rendering time. That's it. At that point it's the Layout/Blocks job to display an HTML on the page depending on the state of your Models.

Skipping The Block

```
$this->loadLayout();
$this->getLayout()->getBlock('root')->setTemplate('yourmodule/get.phtml');

// Expose a variable to the template
// $this->getLayout()->getBlock('root')->setTemplate('recommender/get.phtml')
//            ->assign('r', $r);

$this->renderLayout();
```

Creating Links

When creating links between your controllers it is best to use these helper functions. In admin controllers these are a necessity as admin urls contain a security token and without it your links will redirect back to the admin dashboard.

```
/* Redirect to certain url  */
$this->_redirectUrl($url)

/* Redirect to certain path */
$this->_redirect($path, $arguments=array())

/* Redirect to success page */
$this->_redirectSuccess($defaultUrl)

/* Redirect to error page */
$this->_redirectError($defaultUrl)

/* Set referer url for redirect in response */
$this->_redirectReferer($defaultUrl=null)

/*  Identify referer url via all accepted methods (HTTP_REFERER, regular or base64-encoded request param) */
$this->_getRefererUrl()

/* Check url to be used as internal */
$this->_isUrlInternal($url)
```

There are several special values that effect the url which can be passed in as the second argument.

Here is a quick reference:

Key	Type	Meaning
_absolute	*n/a*	No effect. URLs are always generated as absolute.
_current	bool	Uses the current module, controller, action and parameters
_direct	string	Simply append to the base URL (Uniform Resource Locator), same effect as passing to $routePath. See _store
_escape	bool	Uses &amp; instead of &
_forced_secure	bool	Uses the secure domain given in configuration
_fragment	string	The last part of the URL (Uniform Resource Locator) after a #
_ignore_category	bool	Only applies to `Mage_Catalog_Model_Product_Url::getUrl()`. Prevents category rewrite from being used.
_nosid	bool	Prevents a SID query parameter being used when referencing another store
_query	string or array	If an array it is converted into a string like `?key=value&key=value` which will become the `$_GET` variable.
_secure	bool	Uses the secure domain if allowed in configuration
_store	int or string	Either the numeric store ID or textual store code. It will use the correct domain as the base URL (Uniform Resource Locator).
_store_to_url	bool	Adds ___store to the query parameters. Useful for targetting a store that doesn't have an unique domain.
_type	string	`link` is the default. `direct_link` is useful for bypassing the "store code in URLs" feature. `js`, `media` and `skin` append the domain (and possibly store code) with the relevant directory.
_use_rewrite	bool	Looks up the module/controller/action/parameters in the database for a search engine friendly equivalent.

Overriding a Controller

Replicate the folder structure under your extension's `controllers` folder and copy over the controller you want to modify.

For example if you wanted to override `Mage/CatalogSearch/controllers/ResultController.php` you would create the following:

```
controllers/
└── CatalogSearch
    └── ResultController.php
```

In your `config.xml` add:

```
<config>
    ...
    <frontend>
        <routers>
            <catalogsearch>
                <args>
                    <modules>
                        <yourcompany_yourmodule
before="Mage_CatalogSearch">Yourcompany_Yourmodule_CatalogSearch</yourcompany_yourmodule>
                    </modules>
                </args>
            </catalogsearch>
        </routers>
    </frontend>
    ...
</config>
```

Alter the controller you copied over:

Add: `require_once 'Mage/CatalogSearch/controllers/ResultController.php';` as controlleres don't autoload

Change the class declaration:

```
class Yourcompany_Yourmodule_CatalogSearch_ResultController extends Mage_CatalogSearch_ResultController {
```

Make an intentional PHP syntax error to make sure your controller is getting loaded as the default Mage controller will load if your's is not found.

If you see your error, that is it. Modify the controller to fit your needs.

Creating a Custom Router

Create your controller as normal

In your `config.xml` add:

```
<stores>
    <default>
        <web>
            <routers>
                <yourmodule_custom>
                    <area>frontend</area>
                    <class>Yourcompany_Yourmodule_Controller_Router_Custom</class>
                </yourmodule_custom>
            </routers>
        </web>
    </default>
</stores>
```

In your frontend router section add:

```
// Change to
//<use>standard</use>
<use>yourmodule_custom</use>
```

Under the `code` folder create `Controller` > `Router` > Custom.php

Add the following php and customize it to your custom route.

```
class Yourcompany_Yourmodule_Controller_Router_Custom extends Mage_Core_Controller_Varien_Router_Standard {
    public function match(Zend_Controller_Request_Http $request) {
        $path = explode('/', trim($request->getPathInfo(), '/'));
        // If path doesn't match your module requirements
        if (count($path) > 3 && $path[0] != 'YOURFRONTENDNAME') {
            return false;
        }
        // Define initial values for controller initialization
        $module = $path[0];
        $realModule = 'Yourcompany_Yourmodule';
        $controller = 'index';
        $action = 'index';
        $controllerClassName = $this->_validateControllerClassName(
            $realModule,
            $controller
        );
        // If controller was not found
        if (!$controllerClassName) {
            return false;
        }
        // Instantiate controller class
        $controllerInstance = Mage::getControllerInstance(
            $controllerClassName,
            $request,
            $this->getFront()->getResponse()
        );
        // If action is not found
        if (!$controllerInstance->hasAction($action)) {
            return false; //
        }
        // Set request data
        $request->setModuleName($module);
        $request->setControllerName($controller);
        $request->setActionName($action);
        $request->setControllerModule($realModule);
        // Set your custom request parameter
        $request->setParam('id', $path[2]);
        // dispatch action
        $request->setDispatched(true);
        $controllerInstance->dispatch($action);
        $request->setRouteName('YOURROUTENAME');
        // Indicate that our route was dispatched
        return true;
    }
}
```

Chapter 8

Helpers

Helpers are a way to refactor code into simple helper functions which can be used in *controllers*, *blocks*, and *templates*.

Creating a Helper

Start by declaring your helper in your `config.xml` file.

```
<global>
    <helpers>
        <yourmodule>
            <class>Yourcompany_Yourmodule_Helper</class>
        </yourmodule>
    </helpers>
</global>
```

Under the `Helper` folder create the file, `Data.php`

```
class Yourcompany_Yourmodule_Helper_Data extends Mage_Core_Helper_Abstract {
    public function hello() {
        return 'hello there';
    }
}
```

You can now call your helper by doing:

```
Mage::helper('yourmodule')->hello();
```

Default Helpers

Magento comes with two handy helpers.

- `htmlEscape` - `$this->htmlEscape($comment);`
- `__()` - `$this->__('A sentence you later might want to convert to another language');`

Chapter 9

Events

Introduction

Events allow developers to plug in to key areas of Magento without having to override models/controllers.

Magento raises events in key areas of Magento work flows such as customer creation, product saved, customer login, customer logout and more. An example would be the event 'customer_register_success' which will be raised by Magento immediately after a customer registers to your store.

When to Use Events

If you need to completely change or extend core logic you are usually better off overriding core files. If you just need to add to the core logic we try to use events as they are a little less invasive and tend to break less after Magento upgrades.

Finding Events to Hook on to

Open a terminal and go to the app directory `cd magengo/app`.

To find all events use: `grep -r 'dispatchEvent' *`

To narrow it down by keyword try:

```
grep -r 'dispatchEvent' * | grep 'keyword'
```

Lets say I am looking to create a function that is triggered anytime a person registers for a new account we would search for **register**.

```
grep -r 'dispatchEvent' * | grep register
AccountController.php: Mage::dispatchEvent('customer_register_success',
Invoice.php: Mage::dispatchEvent('sales_order_invoice_register', array($this->_eventObject=>$this, 'order'
=> $order));
CreditmemoController.php: Mage::dispatchEvent('adminhtml_sales_order_creditmemo_register_before', $args);
```

Alternative Way to Find Events to Hook on to

Make sure you have logging turned on.

Open `app/Mage.php`.

In the `dispatchEvent` function add the following line:

```
Mage::log($name);
```

Trigger a front end action you wish to tie to. i.e. add a product to the cart, login, logout, etc.

Take a look in `var/log/system.log` for a list of all events that were fired off.

Don't forget to change `Mage.php` back when you are done.

Create Observer Class

In your module under the folder *Model* create a new file called `Observer.php`.

```
class Yourcompany_Yourmodule_Model_Observer {
    public function foobar($observer) {
        Mage::log('foobar ran!!');
    }
}
```

```
<config>
    ...
    <global>
        <events>
            <customer_register_success>
                <observers>
                    <yourcompany_yourmodule_model_observer>
                        <class>Yourcompany_Yourmodule_Model_Observer</class>
                        <method>foobar</method>
                    </yourcompany_yourmodule_model_observer>
                </observers>
            </customer_register_success>
        </events>
    </global>
    ...
</config>
```

Trigger the event, in this case I will trigger the event by signing up and creating a new user account.

Once you trigger you should see a log entry in the log: `DEBUG (7): foobar ran!!`

Working with Event Objects

Events often contain objects which you can gather information from or trigger a change.

You can tell what objects an event has by looking at the second argument of the `dispatchEvent()` object.

For this example if we were to open `app/code/core/Mage/Customer/AccountController.php` we can see:

```
Mage::dispatchEvent('customer_register_success',
    array('account_controller' => $this, 'customer' => $customer)
);
```

A customer object which we could use to get/set customer data and an account_controller object which we could use to trigger a redirect for example.

If you are unsure what objects your event has just open the file you found in the *Finding Events to Hook on to* section and view the second argument of `dispatchEvent()`.

Creating Custom Events

You can also trigger your own events so that other code you or others write can hook onto them by just calling `dispatchEvent()`.

```
Mage::dispatchEvent('mycustom_event_trigger_name', array('myobject' => $myobject));
```

Chapter 10

Admin Area

Creating a "My Company" Extensions area

Once logged into the admin area of Magento, hover over the `System` menu and click on `Configuration` at the bottom. If you haven't already been in here, this is where a lot of configuration for magento itself is defined, hence the name in the menu.

If your extensions have settings pages, creating a new tab

Adding to your "My Company" area

Add your module into the adminhtml.xml of your "My Company" extension.

```
<?xml version="1.0"?>

<config>
    <sections>
        <mycompany_mymodule translate="label">
            <class>separator-top</class>
            <label>My Extension</label>
            <tab>mycompanytab</tab>
            <sort_order>133</sort_order>
            <show_in_default>1</show_in_default>
            <show_in_website>1</show_in_website>
            <show_in_store>1</show_in_store>
            <groups>
                <general translate="label">
                    <label>My Extension</label>
                    <frontend_type>text</frontend_type>
                    <sort_order>10</sort_order>
                    <show_in_default>1</show_in_default>
                    <show_in_website>1</show_in_website>
                    <show_in_store>1</show_in_store>
                    <fields>
                        ...
                    </fields>
                </general>
            </groups>
        </mycompany_mymodule>
    </sections>
</config>
```

Note: You must log out before you can view your new page in the admin section.

Module Configuration Interface

Types of fields (frontend_type):

You can find the classes for the `fontend_type` in `/lib/Varien/Data/Form/Element/`

Common types:

- Button
- Checkboxes
- Checkbox
- Date
- File
- Hidden
- Imagefile
- Image
- Label
- Link
- Multiline
- Multiselect
- Note
- Obscure
- Password
- Radio
- Radios
- Reset
- Select
- Submit
- Textarea
- Text
- Time

Defining Default Values

This is done in `Yourcompany/Yourmodule/etc/config.xml`:

```
<?xml version="1.0"?>
<config>
  <default>
    <mymodule>
      <general>
        <enabled>1</enabled>
      </general>
    </mymodule>
  </default>
</config>
```

Creating an Admin Controller

Add this to create the link to your new controller in your `config.xml` file:

```
<config>
...
    <adminhtml>
        <menu>
            <sales>
                <children>
                    <yourmodule>
                        <title>Your Module</title>
                        <sort_order>400</sort_order>
                        <action>adminyourmodule/adminhtml_index/index</action>
                    </yourmodule>
                </children>
            </sales>
        </menu>
    </adminhtml>
...
</config>
```

Create the folder, `Adminhtml` in your `controllers` folder

Create a controller, `IndexController.php` in the newly created `Adminhtml` folder.

```
class Yourcompany_Yourmodule_Adminhtml_IndexController extends Mage_Adminhtml_Controller_Action {
    protected function _initAction() {
                $this->loadLayout()->_setActiveMenu('sales')->_addBreadcrumb('sales','sales');
                return $this;
    }
    public function indexAction() {
        $this->_initAction();
        $this->renderLayout();
    }
}
```

Create a route to your controller in your `config.xml` file:

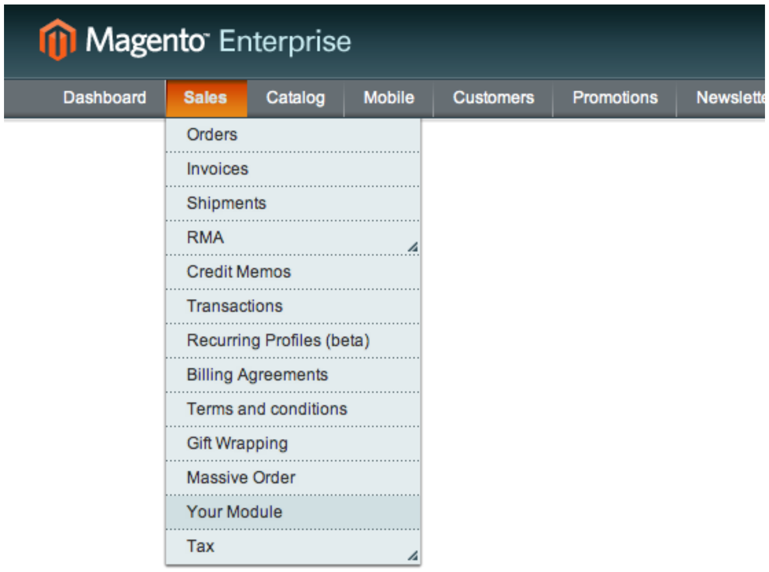
```
<config>
    ...
    <admin>
        <routers>
            <adminyourmodule>
                <use>admin</use>
                <args>
                    <module>Yourcompany_Yourmodule</module>
                    <frontName>adminyourmodule</frontName>
                </args>
            </adminyourmodule>
        </routers>
    </admin>
    ...
</config>
```

Using this method will give you a URL that looks something like `https://yoursite.com/index.php/adminyourmodule/adminhtml_index/index/key/...` plus the admin security key. Which looks a little different than the typical admin URL, which isn't a big deal, it doesn't hurt anything. But if you want your extension to use the same admin URL as the rest of the backend, you can declare your router like so:

```
<config>
  ...
  <admin>
    <routers>
      <adminhtml>
        <args>
          <modules>
            <exports_admin before="Mage_Adminhtml">Yourcompany_Yourmodule_Adminhtml</exports_admin>
          </modules>
          <frontName>adminyourmodule</frontName>
        </args>
      </adminhtml>
    </routers>
  </admin>
  ...
</config>
```

This method will change your URL to `https://yoursite.com/index.php/admin/adminyourmodule/index/key/...`. The way admin pages are called in your extension changes a bit with this method as well, so it is a little more difficult to deal with. If you need the URL to display this way, this is how it's done, otherwise stick with the first approach.

You should now have a menu with a working controller



Adding Javascript

Make sure you have an admin layout in your config

```
<config>
    ...
    <adminhtml>
        <layout>
            <updates>
                <yourcompany_yourmodule>
                    <file>yourcompany/yourmodule.xml</file>
                </yourcompany_yourmodule>
            </updates>
        </layout>
    </adminhtml>
    ...
</config>
```

Create your layout file `yourmodule.xml` in `app/design/adminhtml/default/default/layout/yourcompany/`

In this example we will be adding our JavaScript to the sales order view page. You can change this depending on the urls you want the JavaScript to load on or use `<default>` to specify all of them.

```
<?xml version="1.0"?>
<layout version="1.0.0">
    <adminhtml_sales_order_view>
        <reference name="head">
            <action method="addJs"><script>yourcompany/yourcustomjs.js</script></action>
        </reference>
    </adminhtml_sales_order_view>
</layout>
```

Tip: To make sure your new layout is getting loaded make an intentional xml syntax error by removing `</layout>` for example and you should get an error when reloading any page in the admin backend:

```
Warning: simplexml_load_string(): Entity: line 4: parser error : error parsing attribute nam
```

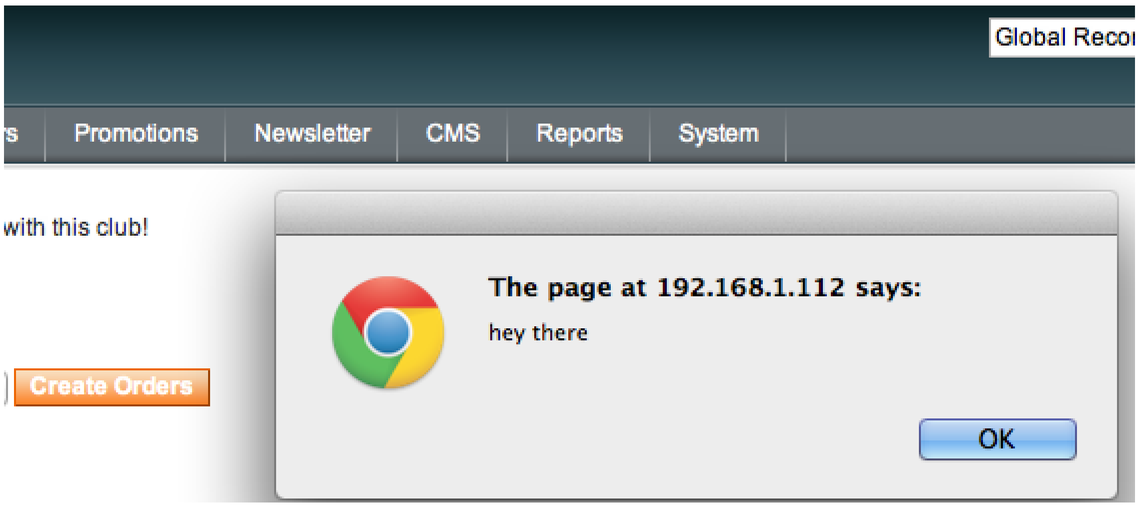
Do not proceed until you see this error as it will eliminate a lot of headache down the road when you are wondering why your JS file is not getting loaded.

At this point Magento should be attempting to load your JavaScript, you can check your console for a missing file error as we haven't added our JavaScript file yet.

Finally create `yourcustomjs.js` in `js/yourcompany`

I am loading jQuery in the backend so mine looks like this:

```
jQuery(document).ready(function($) {
    $('#myButton').click(function() {
        alert('hey there');
    });
});
```



The page at 192.168.1.112 says:

hey there

Chapter 11

Admin Grids

In this chapter we will show you how to create admin grids, and how to use the form builder that will allow for creating and editing the records your grid will be displaying. We will add to the extension created in Chapter 3 and the Model in Chapter 5. The model for this extension will be referenced by `model/table`.

Controller

A good way to start here is to create a subfolder under `controllers` called `Adminhtml`, that way you know the controller contained in this folder is used on the admin portion of your extension. The majority of the folder structure this extension will be using is more or less optional. It helps to keep the file purpose and structure a little clearer, but is not necessary for functionality. Keep in mind, if you choose not to use the folders, the class definitions and some variable values will change.

IndexController.php

```
class Mycompany_Myfirstmodule_Adminhtml_IndexController extends Mage_Adminhtml_Controller_Action{
    public function indexAction(){
            $this->loadLayout();
            $this->_addContent($this->getLayout()->createBlock('myfirstmodule/adminhtml_container'));
            $this->renderLayout();
    }
    public function newAction(){
            $this->_forward('edit');
    }
    public function editAction(){
            $id = $this->getRequest()->getParam('id', null);
        $model = Mage::getModel('model/table');
        if($id){
            $model->load((int) $id);
            if($model->getId()){
                $data = Mage::getSingleton('adminhtml/session')->getFormData(true);
                if($data){
                    $model->setData($data)->setId($id);
                }
            } else {
                Mage::getSingleton('adminhtml/session')->addError('Does not exist');
                $this->_redirect('*/*/');
            }
            Mage::register('data', $model);
        }
            $this->loadLayout();
            $this->_addContent($this->getLayout()->createBlock('myfirstmodule/adminhtml_container_edit'));
        $this->renderLayout();
    }
        public function saveAction(){
                if($data = $this->getRequest()->getPost()){
                $model = Mage::getModel('model/table');
            try {
                    $id = $this->getRequest()->getParam('id');
                    $model->setData($data);
                    Mage::getSingleton('adminhtml/session')->setFormData($data);
                        if($id){ $model->setId($id); }
                    $model->save();
                if(!$model->getId()){
                    Mage::throwException('Error saving record');
                }
                Mage::getSingleton('adminhtml/session')->addSuccess('Record was successfully saved.');
                Mage::getSingleton('adminhtml/session')->setFormData(false);
                                $this->_redirect('*/*/');
            } catch(Exception $e){
                Mage::getSingleton('adminhtml/session')->addError($e->getMessage());
                $this->_redirect('*/*/');
            }
        }
        Mage::getSingleton('adminhtml/session')->addError('No data found to save');
        $this->_redirect('*/*/');
        }
        public function deleteAction(){
                if($id = $this->getRequest()->getParam('id')){
            try{
                $model = Mage::getModel('model/table');
                $model->setId($id);
                $model->delete();
                Mage::getSingleton('adminhtml/session')->addSuccess('The record has been deleted.');
                $this->_redirect('*/*/');
            } catch(Exception $e){
                Mage::getSingleton('adminhtml/session')->addError($e->getMessage());
                $this->_redirect('*/*/edit', array('id' => $id));
            }
        }
        Mage::getSingleton('adminhtml/session')->addError('Unable to find the record to delete.');
        $this->_redirect('*/*/');
        }
}
```

The first function, indexAction, is what puts the grid on the page and should look pretty familiar by now, so we'll move on.

The function newAction simply forwards to editAction. In edit there is a check for the id parameter which, if set, loads the data from that id and registers it with magento. In files to come there are checks to see if this registered data exists to know whether a record is being edited or created.

The save function does its stuff inside the try. The call to the model setData function sets the post data on the model to get ready to save it. If the id parameters exists that means a records is being edited so it sets the id on the model, so it will save the changes to that record instead of adding a new one.

The delete function is pretty similar to save when handling the model, just with no data and calling delete instead of save.

Grid Blocks

These files will be in `Myfirstmodule/Block/Adminhtml`, just so we know these blocks are for the admin.

Container.php

Just as the filename suggests, this file is a container for the grid.

```
class Mycompany_Myfirstmodule_Block_Adminhtml_Container extends Mage_Adminhtml_Block_Widget_Grid_Container{
        public function __construct(){
                $this->_controller = 'adminhtml_container';
                $this->_blockGroup = 'myfirstmodule'; // should be named after the extension
                $this->_headerText = 'Grid Header'; // defines the text for the header of the grid container
                $this->_addButtonLabel = 'Add'; // lets you change the label of the button used to add a
record.
                parent::__construct();
        }
}
```

The first line in the construct method is declaring that this file controls the contents of this container. Meaning there are options set on the container that effect the behavior of the grid. The specified file will be a block, so this value will be the path to the file, which in this case is `Adminhtml/Container.php`. If you opted against using the `Adminhtml` folder it will just be the name of the file.

At the end the parent function is called. Our function is overriding variables used in the parent function, when we're done, the parent function is called to let it run through the rest of its functionality without having to duplicate it.

Grid.php

The next file will be put into a folder named for the container file. In the case of `Myfirstmodule`, this folder will be named `Container`.

```
class Mycompany_Myfirstmodule_Block_Adminhtml_Container_Grid extends Mage_Adminhtml_Block_Widget_Grid{
        public function __construct(){
                parent::__construct();
                $this->setId('containerGrid');
                $this->setDefaultSort('id');
                $this->setDefaultDir('ASC');
        }
        protected function _prepareCollection(){
                $collection = Mage::getModel('model/table')->getCollection();
                $this->setCollection($collection);
                return parent::_prepareCollection();
        }
        protected function _prepareColumns(){
                $this->addColumn('id', array(
                        'header' => 'ID',
                        'align' => 'right',
                        'width' => '50px',
                        'index' => 'id'
                ));
                $this->addColumn('enabled', array(
                        'header' => 'Enabled',
                        'align' => 'left',
                        'sortable' => false,
                        'type' => 'options',
                        'options' => array('No','Yes'),
                        'index' => 'enabled'
                ));
        // Add additional columns

                return parent::_prepareColumns();
        }
    public function getRowUrl($row){
                return $this->getUrl('*/*/edit', array('id' => $row->getId()));
    }
}
```

The construct function defines the id for our grid and sets the field and direction of the default sort for the grid.

The `_prepareCollection` function is where the grid is given the data it is populated with. Depending on what you're using your grid for and where the data is coming from, this could be where some serious magic happens. If you have data with the right models and you're just pulling data in directly from that table, this is pretty much all the code you need.

If you’re table has foreign keys for customers or products that you want the names for this is where you add joins and fields to allow for those fields to display the info you want. Or say your grid is displaying info on files, reports or exports, that are stored on the server. You need to get the list of files and build your collection to pass in, you can build the collection in this function and set it as the collection the grid will use.

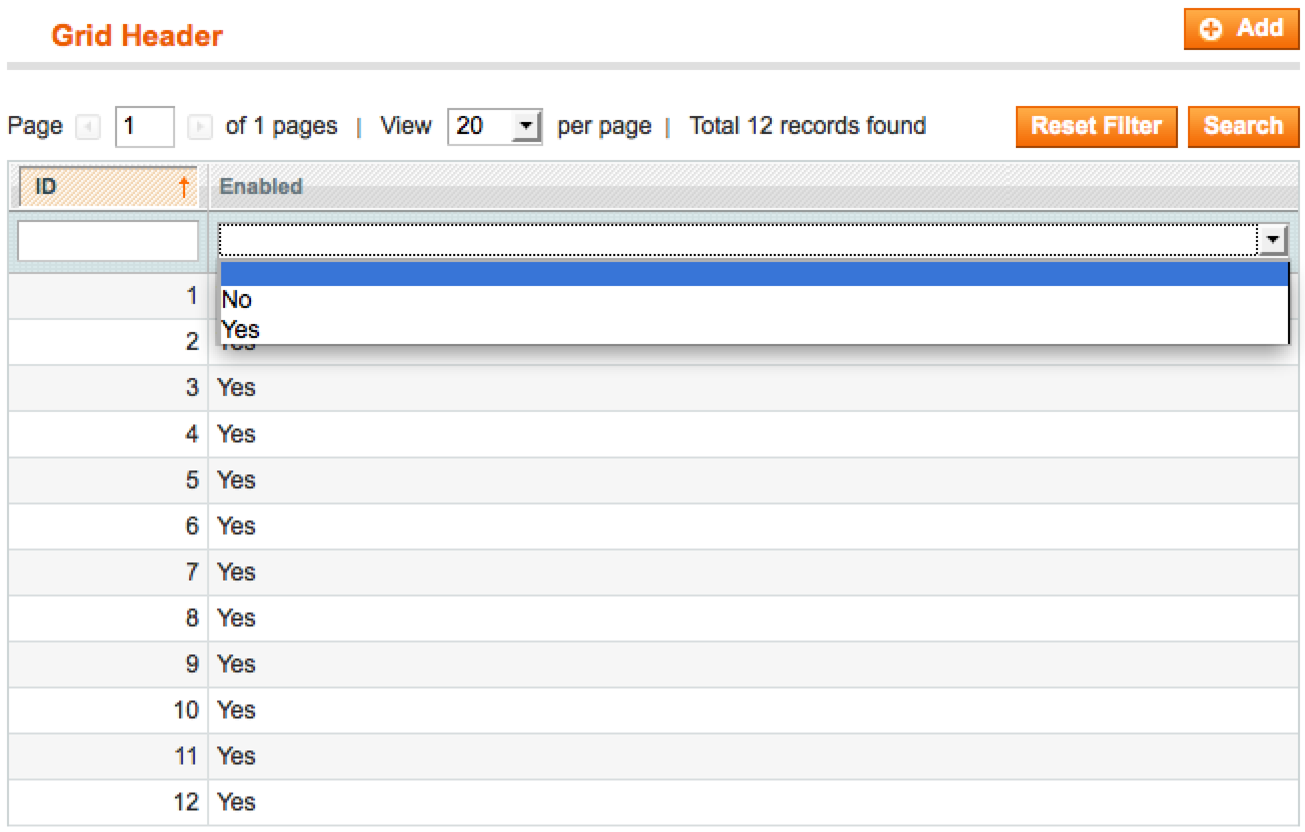
Now we define the columns the grid will display in the `_prepareColumns()` function. The only thing we need to do in here is use `addColumn` for each column and call the parent function. The first parameter for `addColumn()` is a column id, next is an array of params that can define just about anything you would want to change about the column. Header is self explanatory, as well as align. Index is the name of the column in the table your model references that will populate the column of the grid.

Enabled has a couple of extra options. The type options sets the type of filter to be used on the column. Setting it to `options` changes the filter input from text to select, and the options options defines the values of the new filter options filter type.

Two more options of note are `sortable` and `filter`. If you want to turn either of these off on a column you can simply set the value to false. These are on by default and setting filter to true will cause an error.

Lastly we have getRowUrl. This function sets the url for the records in the table. The asterisks in the path tell it to use the same path as the current page. Instead of calling IndexAction in the IndexController, this will call EditAction. The array passes parameters to the edit function, passing the id of the record gives it an id to load from the model so the data can be registered and the form and container will recognize a record is being edited.

If you have correctly setup your new extension and have a menu item that will allow you to reach your extension in the admin you will be able to see a grid like the one below.


Grid Header
Add
Page 1 of 1 pages | View 20 per page | Total 12 records found
Reset Filter
Search
No
Yes

ID	Enabled
1	
2	Yes
3	Yes
4	Yes
5	Yes
6	Yes
7	Yes
8	Yes
9	Yes
10	Yes
11	Yes
12	Yes

Clicking on the add button or a row at this point will give you an error, but that will be resolved with two more files.

Edit.php

This file will sit in the same folder as `Grid.php`.

```
class Mycompany_Myfirstmodule_Block_Adminhtml_Container_Edit extends
Mage_Adminhtml_Block_Widget_Form_Container{
        public function __construct(){
        parent::__construct();
        $this->_objectId = 'id';
        $this->_blockGroup = 'myfirstmodule';
        $this->_controller = 'adminhtml_container';
        $this->_updateButton('save', 'label', 'Save');
                $this->_formScripts[] = "";
    }

        public function getHeaderText(){
        if(Mage::registry('data') && Mage::registry('data')->getId()){
            return 'Edit';
        } else {
            return 'New';
        }
    }
}
```

A lot of the `__construct` method should look familiar. The last 2 lines are the new things here. Update button does exactly what it says it does, you can change the label of the buttons, the onclick events, etc. The formScripts array is really handy, if you want javascript on the page written specifically for the form it can be added to this array. Once added to the array the scripts will be added to the page automatically.

The next function, getHeaderText, is looking for registered data to determine whether to set the form header to edit or new.

Form.php

First create a new folder `Edit`, or whatever you named the previous file.

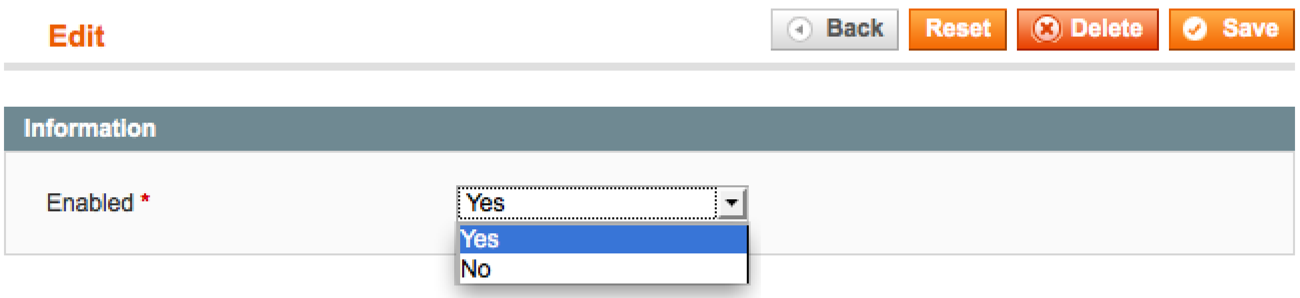
Here we create the fields that are needed to create a new record. A lot of different things can be done on this page, especially when taking advantage of the formScripts array.

```
class Mycompany_Myfirstmodule_Block_Adminhtml_Container_Edit_Form extends Mage_Adminhtml_Block_Widget_Form{
    protected function _prepareForm(){
                $form = new Varien_Data_Form(array(
                        'id' => 'edit_form',
                        'action' => $this->getUrl('*/*/save', array('id' =>
$this->getRequest()->getParam('id'))),
                        'method' => 'post'
                ));
        $form->setUseContainer(true);
              $this->setForm($form);

                $fieldset = $form->addFieldset('form', array('legend' => 'Information'));
                $fieldset->addField('enabled', 'text', array(
                        'label' => 'Enabled',
                        'required' => true,
                        'name' => 'enable'
                ));
        // Add additional fields

                if(Mage::registry('data')){
            $data = Mage::registry('data')->getData();
        } else {
            $data = array();
        }
                $form->setValues($data);
            parent::_prepareForm();
        }
    }
```

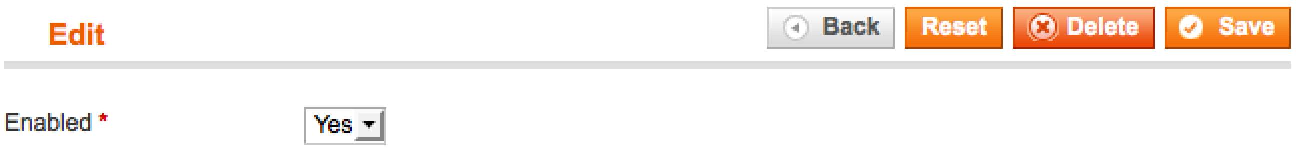
First thing here is to create the form object. You can define the action, method, enctype, etc. After that we are calling setUseContainer, important if you want save button created by the form container to work. The next line is fairly obvious in purpose, but without it the page will only display the header and buttons. So don't miss this line, and make sure the variable is correct if the name changed.

The fields can be added directly to the form, but for a better aesthetic and grouping of fields they are usually added to a fieldset that has been added to the form.

Your form with a fieldset.



Your form without a fieldset.



When adding fields to the fieldset, the first parameter is the id of the field. The id is how the fields will be populated with data. The column name in the table the model is referencing should be the same as this, ideally. The second parameter is the field type. This list has most, if not all, of the possible types:

```
    // text, time, textarea, submit, select, radio, radios, password, obscure, note, multiselect, multiline,
    link, label, image, file, date, checkbox, checkboxes
```

The third parameter is the options array. There are a lot of potential options, depending on what you are doing with the field. Here is a good list of available options:

```
// common - these options can be used on almost all of the field types
'label' => Mage::helper('form')->__('Title'),
'class' => 'required-entry',
'required' => true,
'name' => 'title',
'onclick' => "alert('on click');",
'onchange' => "alert('on change');",
'style' => "border:10px",
'value' => 'hello !!',
'disabled' => false,
'readonly' => true,
'after_element_html' => '<small>Comments</small>',
'tabindex' => 1

// select - two different methods to display select options. the second will give group headings
'values' => array('-1' => 'Please Select..', '1' => 'Option1', '2' => 'Option2', '3' => 'Option3')
'values' => array(
        '-1' => 'Please Select..',
        '1' => array(
                'value' => array(array('value' => '2' , 'label' => 'Option2'), array('value' => '3' ,
'label' =>'Option3')),
                'label' => 'Size'
        ),
        '2' => array(
                'value' => array(array('value' => '4' , 'label' => 'Option4'), array('value' => '5',
'label' => 'Option5')),
                'label' => 'Color'
        ),
)

// radios, checkboxes - this will create groups of the type
'values' => array(
        array('value'=>'1','label'=>'Radio1'),
        array('value'=>'2','label'=>'Radio2'),
        array('value'=>'3','label'=>'Radio3'),
)

// multiselect
'values' => array(
        '-1' => array('label' => 'Please Select..', 'value' => '-1'),
        '1' => array(
                'value' => array(array('value' => '2', 'label' => 'Option2'), array('value' => '3' ,
'label' => 'Option3')),
                'label' => 'Size'
        ),
        '2' => array(
                'value' => array(array('value' => '4', 'label' => 'Option4'), array('value' => '5' ,
'label' => 'Option5')),
                'label' => 'Color'
        ),
)

// link
'href' => 'www.excellencemagentoblog.com'

// date
'format' => Mage::app()->getLocale()->getDateFormat(Mage_Core_Model_Locale::FORMAT_TYPE_SHORT)
```

If you remember back to the controller, the edit function checked if a record was being edited and loaded the data from the model to register it with Magento. Here we check the registry and grab the data if it exists, if not the variable is set to an empty array. Then we set the values for the form that was just created to the data variable. If there was data in the registry, the form will be populated with the data, otherwise the form will be empty.

With that everything should be in place. The index page of your extension should be displaying a grid populated by the data in your model. If you have your model setup but no data, the add button will take you to the form and allow you to save data for the grid to display. Clicking on a row in the grid will take you to the form to edit that record.

Congratulations, you have now built a fully functioning admin grid.

Chapter 12

Working With Teams

Version Control

If this is the first time you are hearing about version control stop what you are doing right now and go research the tremendous benefits you gain by using it. Version control is great for solo developers but as soon as you start working with teams it is an absolute necessity. This is why Joel Spolsky (creator of Stack Overflow) has made this number 1. of this 12 Steps to Better Code.

There are reasons we need to talk about specific strategies for setting up version control with Magento:

- Magento's extension files are scattered in more than one directory
- Versioning the entire Magento directory would be unnecessary as there are loads of temporary files and code you didn't write

One way of combatting these issues is to simply put the entire directory in version control and setup an ignore file to ignore Magento core/temp files. This strategy would be good for a single site but doesn't really allow you to have a seperate repository for individual extensions.

We prefer to put all of our extensions into a single repository and utilize Modman so we can keep all the files pertaining to a specific extension all in one place. This allows us to deploy that extension into multiple sites and update the extension across all sites with a single command.

We use Bitbucket as it provides free private repos and supports both *Mercurial* and *git*.

Modman

Modman stands for **Mod**ule **Man**ager and is a great script built by Colin Mollenhour which can be found at: github.com/colinmollenhour/modman

From the project description:

"Developing extensions for software that doesn't allow you to separate your files from core files, and keeping that extension under version control and making it easy to deploy is now much, much easier. Development of this script was inspired by Magento which forces you to mix your extension files all throughout the core code directories. With modman, you can specify in a text file where you want your directories and files to be mapped to, and it will maintain symlinks for you so that your code is easy to hack and deploy."

Modman allows us to keep our extensions seperate from Magento and simply link the files to the locations Magento expects them to be. This makes version control easy, deployment to multiple magento installations easy, and also keeps the files in a single easy to find folder.

Chapter 13

Bonus Chapters

Chapter 14

Creating a Custom Extension to Test Transactional Emails

In this tutorial we will build an extension from scratch with an admin backend which we can use to:

- Demonstrate the ability to send transactional emails from a custom module
- Use this module to easily test customizations to transactional emails

If you do any email template customizations at all the time you will save using this module/technique will pay for this book 10 times over.

Initializing your Extension

By now you should know the drill:

Under `Mycompany` folder create a new extension called `Transactionaltest`

```
app/
└── code
    └── local
        └── Mycompany
            └── Transactionaltest
```

Build your extension folder structure:

```
app/
└── code
    └── local
        └── Mycompany
            └── Transactionaltest
                ├── Block
                ├── controllers
                ├── etc
                ├── Helper
                ├── Model
                └── sql
```

Add in your `config.xml` file:

```
<?xml version="1.0"?>
<config>
    <modules>
        <Mycompany_Transactionaltest>
            <version>0.1.0</version>
        </Mycompany_Transactionaltest>
    </modules>
</config>
```

Create your module file: `Mycompany_Transactionaltest.xml` under `app/etc/modules`:

```
app/
├── code
|   └── local
|       └── Mycompany
|           └── Transactionaltest
|               ├── Block
|               ├── controllers
|               ├── etc
|               |   └── config.xml
|               ├── Helper
|               ├── Model
|               └── sql
└── etc
    └── modules
        └── Mycompany_Transactionaltest.xml
```

```
<?xml version="1.0"?>
<config>
    <modules>
        <Mycompany_Transactionaltest>
            <active>true</active>
            <codePool>local</codePool>
        </Mycompany_Transactionaltest>
    </modules>
</config>
```

To make sure your module is initialized properly go to: **System** > **Configuration** > **Advanced**

Create an Admin Backend

Create a `system.xml` in your `etc` folder:

```
<?xml version="1.0"?>
<config>
    <sections>
        <mycompany_transactionaltest translate="label">
```

```
                <class>separator-top</class>
                <label>Test Transactional Emails</label>
                <tab>advanced</tab>
                <sort_order>133</sort_order>
                <show_in_default>1</show_in_default>
                <show_in_website>1</show_in_website>
                <show_in_store>1</show_in_store>
                <groups>
                    <general translate="label">
                        <label>Transactional Test</label>
                        <frontend_type>text</frontend_type>
                        <sort_order>10</sort_order>
                        <show_in_default>1</show_in_default>
                        <show_in_website>1</show_in_website>
                        <show_in_store>1</show_in_store>
                        <fields>
                            <enabled translate="label">
                                <label>Enabled</label>
                                <frontend_type>select</frontend_type>
                                <source_model>adminhtml/system_config_source_yesno</source_model>
                                <sort_order>1</sort_order>
                                <show_in_default>1</show_in_default>
                                <show_in_website>1</show_in_website>
                                <show_in_store>1</show_in_store>
                            </enabled>
                        </fields>
                    </general>
                </groups>
            </mycompany_transactionaltest>
        </sections>
    </config>
```

Here we are adding a link under the *ADVANCED* section on the side navigation indicated by the `<tab>advanced</tab>` section.

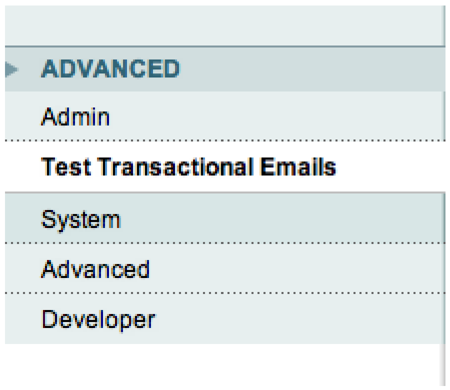
If you go to the backend admin section now you should see the link but it will currently display a 404 page as we haven't given access to it yet.

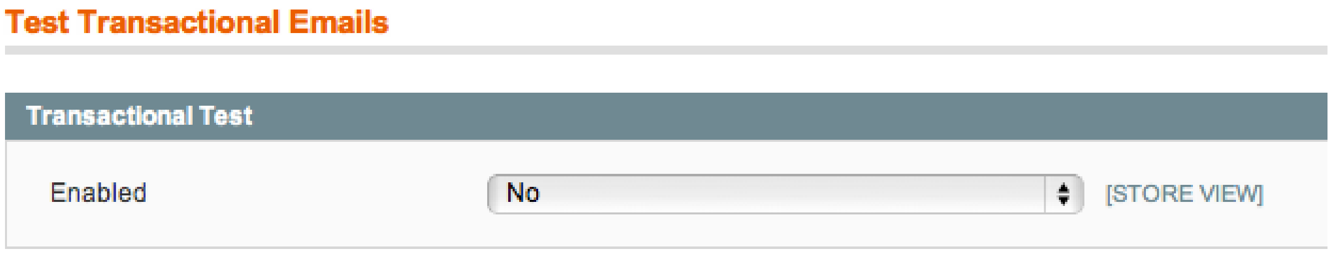
Create an `adminhtml.xml` file under the `etc` folder:

```
<?xml version="1.0"?>
<config>
  <acl>
    <resources>
      <all>
        <title>Allow Everything</title>
      </all>
      <admin>
        <children>
          <system>
```

```
                <children>
                  <config>
                    <children>
                      <mycompany_transactionaltest translate="title">
                        <title>Transactional Test</title>
                        <sort_order>101</sort_order>
                      </mycompany_transactionaltest>
                    </children>
                  </config>
                </children>
              </system>
            </children>
          </admin>
        </resources>
      </acl>
    </config>
```

Make sure to log out to refresh the permissions and you should now be able to view your newly created admin page.




Test Transactional Emails

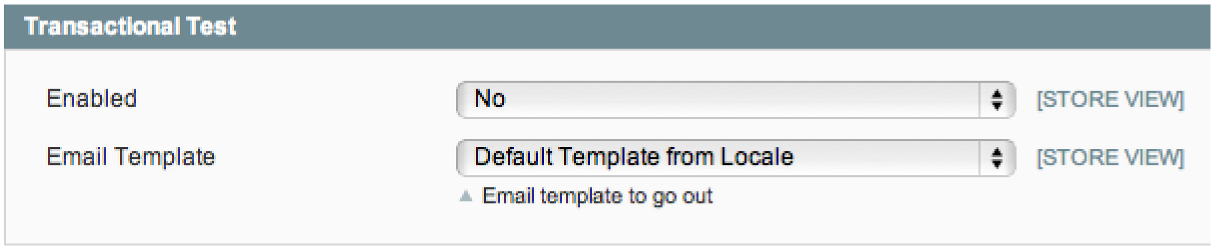
Transactional Test

Enabled No [STORE VIEW]

Now we are going to want a dropdown of all available transactional email templates. Luckily Magento does this already in the backend under *Sales Emails* so we can model after that.

In your `system.xml` file in the `<fields>` tag add in:

```
<email_template translate="label">
    <label>Email Template</label>
    <frontend_type>select</frontend_type>
    <source_model>adminhtml/system_config_source_email_template</source_model>
    <sort_order>6</sort_order>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>1</show_in_store>
    <comment>Email template to go out</comment>
</email_template>
```



Transactional Test

Enabled: No [STORE VIEW]

Email Template: Default Template from Locale [STORE VIEW]

Email template to go out

Create a Controller

Under the `controllers` folder add the file `IndexController.php`

```
class Mycompany_Transactionaltest_IndexController extends Mage_Core_Controller_Front_Action {
    public function testAction(){
        echo 'can you see this';
    }
}
```

Note: I typically start out all my controllers like this so I can tell if my config is getting loaded properly.

And add the router to your `config.xml` file:

```
<?xml version="1.0"?>
<config>
    ...
    <frontend>
        <routers>
            <transactionaltest>
                <use>standard</use>
                <args>
                    <module>Mycompany_Transactionaltest</module>
                    <frontName>transactionaltest</frontName>
                </args>
            </transactionaltest>
        </routers>
    </frontend>
    ...
</config>
```

Once you verified your controller is working and you can see the text *can you see this* you can swap out the code for something a little more productive:

```
class Mycompany_Transactionaltest_IndexController extends Mage_Core_Controller_Front_Action {
    public function testAction(){
        $customer_id = 6; // A valid customer id from the backend
        $customer = Mage::getModel('customer/customer')->load($customer_id);
        $customer_data = $customer->getData();
        $full_name = $customer_data['firstname']. ' '. $customer_data['lastname'];
        $to_email = $customer_data['email'];

        // Pass in variables you want in your email template.
        // If you are testing an email regaring an order like (new-order)
        // you can load an order object and pass it in here.
        $email_variables = array();
        $email_variables['customer'] = $customer;
        //$email_variables['order'] = Mage::getModel('sales/order')->load($yourorderid);
        $email_variables['subscriber'] = 'foobar';
        $email_variables['yourvariable'] = 'whateveryouwant';

        $email = Mage::getModel('core/email_template');
        $email->sendTransactional(
            // This line corresponds and match the tags in system.xml
            // It returns the id of the email template selected in the admin backend
            (integer) Mage::getStoreConfig('mycompany_transactionaltest/general/email_template'),

            'general', // From Address: general, sales, support, custom1, custom2
            $to_email,
            $full_name,
            $email_variables
        );
    }
}
```

Thats it, change the `$customer_id` to a customer that you made with a valid email address, select an email template in the backend and hit the url for your contoller to fire off the email.

Not only do you now know how to send transactional emails from an extension, you also have a very useful extension for testing different email templates.

Chapter 15
A Tool For Automatically Creating Parts of your App

Generating all those files and folders necessary to start building your extension can be a pain.

```
php magentomatic.php create
```

Follow the prompts

Chapter 16

Magentomatic

Restore Admin Access

From the magentomatic directory run, `php magentomatic.php password`

You can now log in with the account:

- username: **magentomatic**
- password: **magentomatic**

Remember to change your password after logging in.

Backup Database

From the magentomatic directory run `php magentomatic.php backup`

and follow the prompts to make a backup of the database.

Restore Database

WARNING: This will write to your Database, use caution when using this.

From the magentomatic directory run `php magentomatic.php restore`

and follow the prompts to restore.

Config Profiles

Magentomatic has a great way of automatically setting and saving Magento configuration settings.

To load a config profile from the magentomatic `profiles` directory run: `php magentomatic.php config`

To save your current config profile run: `php magentomatic.php config myconfig`

Note: Fields that are encrypted (like payment gateway keys) will not import correctly, you will need to set these values by hand in the backend as you normally would. This is a security feature.

Saved profiles can be edited they follow ini syntax:

```
design/head/demonotice = "1"
web/unsecure/base_url = "http://192.168.1.112/magento/"
web/secure/base_url = "http://192.168.1.112/magento/"
design/package/name = "enterprise"
admin/security/session_cookie_lifetime = "86400"
dev/template/allow_symlink = 1
# Example of a comment
admin/url/custom = "http://192.168.1.112/magento/admin/"
```

You can see your current default paths and values by viewing Magento's `core_config_data` table.

Clear Cache

From the magentomatic directory run `php magentomatic.php clear_cache`

Chapter 17

Miscellaneous

Cron Jobs

Cron jobs are used when you want to run a portion of your code periodically. As with all time sensitive code make sure your server time is up to date.

General XML syntax for setting up a cron in your extensions `config.xml` file:

```
<config>
...
    <crontab>
        <jobs>
            <yourcompany_yourmodule>
                <schedule>
                    <cron_expr>0,15,30,45 * * * *</cron_expr>
                </schedule>
                <run>
                    <model>yourmodule/model::method</model>
                </run>
            </yourcompany_yourmodule>
        </jobs>
    </crontab>
...
</config>
```

To run a function from your observer you can set `<model>` to `yourmodule/observer::yourfunction`

Note: If your extension has no models and you are trying to run a function from your observer (`Observer.php`) you will need to make sure you have declared a model in your `config.xml`:

```
<models>
    <yourmodule>
        <class>Yourcompany_Yourmodule_Model</class>
    </yourmodule>
</models>
```

Verify the Cron Job

Hit the the cron url at *http://yourmagento/cron.php* or alternatively hit it from the command line `php cron.php`.

With your favorite mysql tool open up the `yourprefixifyouhaveone_cron_schedule` table and verify that `yourcompany_yourmodule` is listed in the `job_code` column.

If you look at the `scheduled_at` column you can see when the next time your job is scheduled to run.

You may not see your job listed if it is set further in the future than Magento's *Schedule Ahead for* config setting in **System** > **Configuration** > **System** > **Cron**.

Default Cron Jobs in Magento

You can get a good list of default cron jobs and the methods they call by running this command `find . -name "*.xml" | xargs -L10 grep -A4 "cron_expr"` in the root Magento directory.

Logging

Magento comes with a built-in logging which can be enabled in the back-end under **System** > **Configuration** > **Developer**. Make sure your `var/log/` folder is writable.

Examples

```
Mage::log('This will show up in system.log');
Mage::log('My log variable: '.$myVariable);
Mage::log($myArray);
Mage::log($myObject);
Mage::log($myObject->debug());
Mage::logException('this will show up in exception.log');
```

Logging to your own file:

```
Mage::log('My own log file', null, 'mylogfile.log');
// Log entries will be in their own file at var/log/mylogfile.log.
```

Making Sense of Magento's Naming Conventions

Whenever you are browsing the code you will notice calls to something like `module/some_thing`. This string maps to a class name.

Examples:

```
$product = Mage::getModel('customer/address');
//Translates into Mage_Customer_Model_Address

$url = $this->helper('customer')->getLoginPostUrl();
//Translates into Mage_Customer_Helper_Data ("/data" is appended by default to helpers)

$url = $this->helper('giftmessage/url')->getSaveUrl();
//Translates into Mage_Giftmessage_Helper_Url

<block type="catalog/product_list" name="product_list" template="catalog/product/list.phtml" />
<!-- Even XML translates into Mage_Catalog_Block_Product_List -->
```

Site-wide Notices

Site-wide notices are saved in the session and displayed on the next page load. Since the customer session is seperate from the admin session we need to use the proper session object:

- Front end: `Mage::getSingleton('core/session')`
- Backend: `Mage::getSingleton('adminhtml/session')`

Success

```
Mage::getSingleton('core/session')->addSuccess('Successfully created payment option.');
```

Error

```
// Example with translate capabilities
Mage::getSingleton('core/session')->addError($this->__('An error occurred while adding item to wishlist.'));
```

Notice

```
Mage::getSingleton('core/session')->addNotice($notice);
```

Warning

```
Mage::getSingleton('core/session')->addWarning("You are not allowed to add more than ($this->max_items)
items in your cart.");
```

Client-side Form Validation

Magento has built in JS form validation provided by the Prototype library:

First you need to create a Form (form.js) object to represent your form.

```
<script type="text/javascript">
//< ![CDATA[
  var myForm= new varienForm('formId', true);
//]]>
</script>

<form id="formId">
    <!-- Just change class to validation type below (you can have multiple) -->
    <input type="text" name="whatever" class="required-entry" />
</form>
```

- **validate-select** Please select an option.
- **required-entry** This is a required field.
- **validate-number** Please enter a valid number in this field.
- **validate-digits** Please use numbers only in this field. please avoid spaces or other characters such as dots or commas.
- **validate-alpha** Please use letters only (a-z or A-Z) in this field.
- **validate-code** Please use only letters (a-z), numbers (0-9) or underscore(_) in this field, first character should be a letter.
- **validate-alphanum** Please use only letters (a-z or A-Z) or numbers (0-9) only in this field. No spaces or other characters are allowed.
- **validate-street** Please use only letters (a-z or A-Z) or numbers (0-9) or spaces and # only in this field.
- **validate-phoneStrict** Please enter a valid phone number. For example (123) 456-7890 or 123-456-7890.
- **validate-phoneLax** Please enter a valid phone number. For example (123) 456-7890 or 123-456-7890.
- **validate-fax** Please enter a valid fax number. For example (123) 456-7890 or 123-456-7890.
- **validate-date** Please enter a valid date.
- **validate-email** Please enter a valid email address. For example johndoe@domain.com.
- **validate-emailSender** Please use only letters (a-z or A-Z), numbers (0-9) , underscore(_) or spaces in this field.
- **validate-password** Please enter 6 or more characters. Leading or trailing spaces will be ignored.

- **validate-admin-password** Please enter 7 or more characters. Password should contain both numeric and alphabetic characters.
- **validate-cpassword** Please make sure your passwords match.
- **validate-url** Please enter a valid URL. http:// is required
- **validate-clean-url** Please enter a valid URL. For example http://www.example.com or www.example.com
- **validate-identifier** Please enter a valid Identifier. For example example-page, example-page.html or anotherlevel/example-page
- **validate-xml-identifier** Please enter a valid XML-identifier. For example something_1, block5, id-4
- **validate-ssn** Please enter a valid social security number. For example 123-45-6789.
- **validate-zip** Please enter a valid zip code. For example 90602 or 90602-1234.
- **validate-date-au** Please use this date format: dd/mm/yyyy. For example 17/03/2006 for the 17th of March, 2006.
- **validate-currency-dollar** Please enter a valid $ amount. For example $100.00.
- **validate-one-required** Please select one of the above options.
- **validate-one-required-by-name** Please select one of the options.
- **validate-not-negative-number** Please enter a valid number in this field.
- **validate-state** Please select State/Province.
- **validate-new-password** Please enter 6 or more characters. Leading or trailing spaces will be ignored.
- **validate-greater-than-zero** Please enter a number greater than 0 in this field.
- **validate-zero-or-greater** Please enter a number 0 or greater in this field.
- **validate-cc-number** Please enter a valid credit card number.
- **validate-cc-type** Credit card number doesn't match credit card type
- **validate-cc-type-select** Card type doesn't match credit card number
- **validate-cc-exp** Incorrect credit card expiration date
- **validate-cc-cvn** Please enter a valid credit card verification number.
- **validate-data** Please use only letters (a-z or A-Z), numbers (0-9) or underscore(_) in this field, first character should be a letter.
- **validate-css-length** Please input a valid CSS-length. For example 100px or 77pt or 20em or .5ex or 50%
- **validate-length** Maximum length exceeded.

Accessing Magento Functions Outside of Magento

Create a file in the root directory and load the mage app like this:

```
// This line is all you need
require_once 'app/Mage.php'; umask(0); Mage::app('default');

// Example loading a category
$category = Mage::getModel('catalog/category')->load(1);
print_r($category->getData());
```

Magento's Image Resize Helper

- `->constrainOnly(true)` This will not resize an image that is smaller than the dimensions inside the resize() part.
- `->keepAspectRatio(true)` This will not distort the height/width of the image.
- `->keepFrame(false)` This will not put a white frame around your image.

```
echo $this->helper('catalog/image')->init($_product, 'image')
    ->constrainOnly(true)
    ->keepAspectRatio(true)
    ->keepFrame(false)
    ->resize(350, null);
```

Add Column to Sales Order Table

1. Create a new module with own setup class extended from `Mage_Sales_Model_Mysql4_Setup`

2. Your config

```
<global>
    <resources>
        <your_module_setup>
            <setup>
                <module>Your_Module</module>
                <class>Mage_Sales_Model_Mysql4_Setup</class>
            </setup>
        </your_module_setup>
    </resources>
</global>
```

1. Installer

```
$installer = $this;
$installer->startSetup();
$installer->addAttribute(
    'order',
    'your_attribute_code',
    array(
        'type' => 'int', /* varchar, text, decimal, datetime */,
        'grid' => false /* or true if you wan't use this attribute on orders grid page */
    )
);
$installer->endSetup();
```

Chapter 18

Recipes

General

Writing to the log

```
Mage::log("nice to learn this ");
Mage::log($myObject);
```

Log to custom file

```
Mage::log('Your Log Message', Zend_Log::INFO, 'your_log_file.log');
```

Debug using Zend

```
echo Zend_Debug::dump($thing_to_debug, 'debug');
```

Get Magento Store ID

```
Mage::app()->getStore()->getStoreId();
```

Get Current Url

```
echo Mage::helper('core/url')->getCurrentUrl();
```

Get Base Paths

```
$base_path = Mage::getBaseDir('base');
var_dump($base_path);

$etc_path = Mage::getBaseDir('etc');
var_dump($etc_path);
```

Options:

- app_dir
- base_dir
- code_dir
- design_dir
- etc_dir
- lib_dir

- locale_dir
- media_dir
- skin_dir
- var_dir
- tmp_dir
- cache_dir
- log_dir
- session_dir
- upload_dir
- export_dir

Get Magento Media Url

```
Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_LINK);
```

Get Magento Media Url

```
Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_MEDIA);
```

Get Magento Skin Url

```
Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_SKIN);
```

Get Magento Store Url

```
Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_WEB);
```

Get Magento Js Url

```
Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_JS);
```

Get Secure Url

```
Mage::getUrl('',array('_secure'=>true)
Mage::getUrl('',array('_forced_secure'=>true)
```

Accessing a helper

```
Mage::helper('module/data')->functionName();
```

Format a price value?

```
Mage::helper('core')->formatPrice($amount);
```

Get an Array of Country names

```
$countryList = Mage::getResourceModel('directory/country_collection')
    ->loadData()
    ->toOptionArray(false);
echo '<pre>';
print_r( $countryList);
exit('</pre>');
```

Create a country dropdown

```
$_countries = Mage::getResourceModel('directory/country_collection')
    ->loadData()
    ->toOptionArray(false);
if(count($_countries) > 0){ ?>
    <select name="country" id="country">
        <option value="">-- Please Select --</option>
        <?php foreach($_countries as $_country){ ?>
            <option value="<?php echo $_country['value']; ?>">
                <?php echo $_country['label']; ?>
            </option>
        <?php } ?>
    </select>
<?php } ?>
```

Create state dropdown for use in forms

```
$regionCollection = Mage::getModel('directory/region')
    ->getCollection()
    ->addCountryFilter('US');
$regions = $regionCollection->toOptionArray();
$general->addField('state', 'select', array(
    'name'      => 'state',
    'label'     => Mage::helper('core')->__('Location State'),
    'value'     => 1,
    'values'    => $regions,
    'required'  => true
));
```

All Store Views?

```
Mage::app()->getStores(); // pass true to include the admin store view, too
```

How do I get an instance of an attribute?

```
Mage::getSingleton('eav/config')->getAttribute($entityType, $attributeCode)
```

What if the attribute is a dropdown?

```
$product->getAttributeText('brandkey')
```

Price rounding

```
echo Mage::getStoreConfig('general/store_information/name');
echo Mage::getModel('sales/order')->formatPricePrecision($_product->getFinalPrice(), 3);
```

Requests

How do I get at GET/POST parameters?

```
Mage::app()->getRequest()->getParam('param_name'); // single parameter
Mage::app()->getRequest()->getParams(); // all parameters
```

Session

Get Customer Session

```
$customer = Mage::getSingleton('customer/session')->getCustomer();
```

Setting Session Variables

```
Mage::getSingleton('core/session')->setBlahBlahBlah('my data');
$myData = Mage::getSingleton('core/session')->getBlahBlahBlah();
```

Next Page Notifications

```
Mage::getSingleton('core/session')->addError('Custom error here');
Mage::getSingleton('core/session')->addWarning('Custom warning here');
Mage::getSingleton('core/session')->addNotice('Custom notice here');
Mage::getSingleton('core/session')->addSuccess('Custom success here');
```

Customer

Customer Model

```
$customer = Mage::getModel('customer/customer');
```

Get Email address by customer id

```
$customer = Mage::getModel('customer/customer');
$customer->load(10);
```

```
$data = $customer->getData();
echo $data['email'];
```

Get Customer Birthday

```
$customer = Mage::getSingleton('customer/session')->getCustomer();
echo 'birthday: '. $customer->getData('dob');
```

Find customers by created_at date:

```
$customer = Mage::getModel('customer/customer')->getResourceCollection();
$customer->addFieldToFilter('created_at', array('date' => true, 'to' => '2012-01-09 23:30:43'));
$data = $customer->getData();
print_r($data);
```

Get Customer Group Id

```
$customer = Mage::getSingleton('customer/session')->getCustomer();
echo $customer->getGroupId();
```

Get Customer Firstname

```
$customer->getName();
$customer->getFirstname();
```

Detect if Customer is Logged in

```
$this->helper('customer')->isLoggedIn()
```

Products

Product Model

```
$product  = Mage::getModel('catalog/product');
```

Load by Sku

```
$product = Mage::getModel('catalog/product')->loadByAttribute('sku', $sku);
```

Categories

```
$categories = $product->getCategoryCollection();
```

Load Category by ID

```
$category = Mage::getModel('catalog/category')->load(3);
```

Get Category Name

```
$category->getName();
```

Get Category URl

```
$category->getUrl();
```

Is product purchasable?

```
if($_product->isSaleable()) { // do stuff }
```

Get Products by Category ID

```
$category = Mage::getModel('catalog/category')->load(5);
$products = $category->getProductCollection();
print_r($products->getData());
```

Product stock quantity

```
$qtyStock = Mage::getModel('cataloginventory/stock_item')->loadByProduct($_product)->getQty();
```

Get all products from an array of product ids

```
$products = Mage::getModel('catalog/product')->getCollection()->addAttributeToFilter('entity_id',
array('in' => $productIds));
```

Get the configurable/grouped/bundled product a simple product belongs to

```
$simpleProduct->loadParentProductIds();
$parentProductIds = $simpleProduct->getParentProductIds();
```

Get the simple products assigned to a configurable product

```
$configProduct->getTypeInstance()->getUsedProductCollection($configProduct);
```

Get Products by type

```
$collectionSimple = Mage::getResourceModel('catalog/product_collection')
                ->addAttributeToFilter('type_id', array('eq' => 'simple'));
```

Get Attribute Set Id

```
$product->getAttributeSetId();
```

Get Attribute Set Name

```
$attributeSetName = Mage::getModel('eav/
entity_attribute_set')->load($_product->getAttributeSetId())->getAttributeSetName();
```

Orders

Order Model

```
$order = Mage::getModel('sales/order');

$customerOrderCollection = Mage::getModel('sales/order')->getCollection();
$customerOrderCollection->addFieldToFilter('customer_id', $customer_id);
```

Get Order Status

```
$order->getStatus();
```

Add Order Status History Comment

```
$comment = 'blah blah';
$order->addStatusHistoryComment($comment);
$order->save();
```

Load by Increment ID

```
$orderIncrementId = YOUR_ORDER_INCREMENT_ID;
$order = Mage::getModel('sales/order')
                ->loadByIncrementId($orderIncrementId);
```

Set Custom Order Status

```
// Custom status must exist
$order = $model->load($order_id);
            $order->setStatus('delivered');
            $order->save();

// change order status to 'Pending'
$order->setState(Mage_Sales_Model_Order::STATE_NEW, true)->save();

// change order status to 'Pending Paypal'
$order->setState(Mage_Sales_Model_Order::STATE_PENDING_PAYMENT, true)->save();

// change order status to 'Processing'
$order->setState(Mage_Sales_Model_Order::STATE_PROCESSING, true)->save();

// change order status to 'Completed'
```

```
$order->setState(Mage_Sales_Model_Order::STATE_COMPLETE, true)->save();

// change order status to 'Closed'
$order->setState(Mage_Sales_Model_Order::STATE_CLOSED, true)->save();

// change order status to 'Canceled'
$order->setState(Mage_Sales_Model_Order::STATE_CANCELED, true)->save();

// change order status to 'Holded'
$order->setState(Mage_Sales_Model_Order::STATE_HOLDED, true)->save();
```

Invoices

Get Invoice by ID

```
$invoice = Mage::getModel('sales/order_invoice')->load($invoiceId);
```

Capture invoice

```
if($invoice->canCapture()){
    $invoice->capture();
}

if($order->canInvoice()){
    $invoice = $order->prepareInvoice();
    $invoice->register();

    // Send email
    $invoice->setEmailSent(true);
    $invoice->getOrder()->setIsInProcess(true);
    $transactionSave = Mage::getModel('core/resource_transaction')
        ->addObject($invoice)
        ->addObject($invoice->getOrder())
        ->save();
}
```

Shipments

Get all tracking numbers from shipments

```
$model = Mage::getModel('sales/order_shipment');
$tracks = $model->load(1)->getAllTracks();
//$collection = $model->getCollection();
//$data = $collection->getData();
foreach ($tracks as $track) {
    echo $track->getNumber();
}
```

Shipping info for magento

```
$_order = $this->getShipment()->getOrder();
$_shippingAddress = $_order->getShippingAddress();
echo $_shippingAddress->getFirstname() . '<br />';
echo $_shippingAddress->getLastname() . '<br />';
echo $_shippingAddress->getCompany() . '<br />';
echo $_shippingAddress->getStreetFull() . '<br />';
echo $_shippingAddress->getRegion() . '<br />';
echo $_shippingAddress->getCity() . '<br />';
echo $_shippingAddress->getPostcode() . '<br />';
echo $_shippingAddress->getCountry_id() . '<br />';
```

Cart

Magento Cart Total

```
echo $this->helper('checkout')->formatPrice(Mage::getSingleton('checkout/
cart')->getQuote()->getGrandTotal());
```

Blocks

Call Static Block

```
echo $this->getLayout()->createBlock('cms/block')->setBlockId('block-name')->toHtml();
```

Admin

Permissions

```
/*
// Example code to get all the available permissions
$resources = Mage::getModel('admin/roles')->getResourcesTree();
$nodes = $resources->xpath('//*[@aclpath]');
echo '<dl>';
foreach($nodes as $node){
    echo '<dt>' . (string)$node->title . '</dt>';
    echo '<dd>' . $node->getAttribute('aclpath') . '</dd>';
}
echo '</dl>';
*/

var_dump(Mage::getSingleton('admin/session')->isAllowed('admin/sales/order/actions/scforce'));
```

Generate a link from an admin page

```
echo Mage::helper("adminhtml")->getUrl("mymodule/adminhtml_index/action/");
```

Programmatically change config data

```
// find 'path' in table 'core_config_data' e.g. 'design/head/demonotice'
$my_change_config = new Mage_Core_Model_Config();
// turns notice on
$my_change_config->saveConfig('design/head/demonotice', "1", 'default', 0);
// turns notice off
$my_change_config->saveConfig('design/head/demonotice', "0", 'default', 0);
```

Get Config Valus

```
$value = Mage::getStoreConfig('[MODULE]/[SECTION]/[FIELD]', $storeId
```

Common Config Values

```
//General contact
Mage::getStoreConfig('trans_email/ident_gerneral/name');
Mage::getStoreConfig('trans_email/ident_gerneral/email');

//Sales Representative
Mage::getStoreConfig('trans_email/ident_sales/name');
Mage::getStoreConfig('trans_email/ident_sales/email');

//Customer Support
Mage::getStoreConfig('trans_email/ident_support/name');
Mage::getStoreConfig('trans_email/ident_support/email');

//Custom email1
Mage::getStoreConfig('trans_email/ident_custom1/name');
Mage::getStoreConfig('trans_email/ident_custom1/email');

//Custom email2
Mage::getStoreConfig('trans_email/ident_custom2/name');
Mage::getStoreConfig('trans_email/ident_custom2/email');
```

Events

```
Mage::dispatchEvent($eventName);
```

Enterprise

Get Reward balance (enterprise)

```
$customer = Mage::getSingleton('customer/session')->getCustomer();
$reward = Mage::getModel('enterprise_reward/reward')
            ->setCustomer($customer)
            ->setWebsiteId(Mage::app()->getWebsite()->getId())
            ->loadByCustomer();

$balance = $reward->getPointsBalance();
echo $balance;
```

Miscellaneous

Connect directly to a table in Magento

```
$w = Mage::getSingleton('core/resource')->getConnection('core_write');
$result = $w->query('select 'entity_id' from 'catalog_product_entity'');

if (!$result) {
return false;
}

$row = $result->fetch(PDO::FETCH_ASSOC);
if (!$row) {
return false;
}
```

Get the MySQL table name of any Magento Object

```
$r = Mage::getResourceSingleton('core/resource')->getConnection('core_read');
$tableName = $r->getTable('catalog/product');
```

Documentation for Magento Developers

Mark Sanborn, Mark Lehm, Ryan Leising, and Richard Fowler

Tweet about the book!