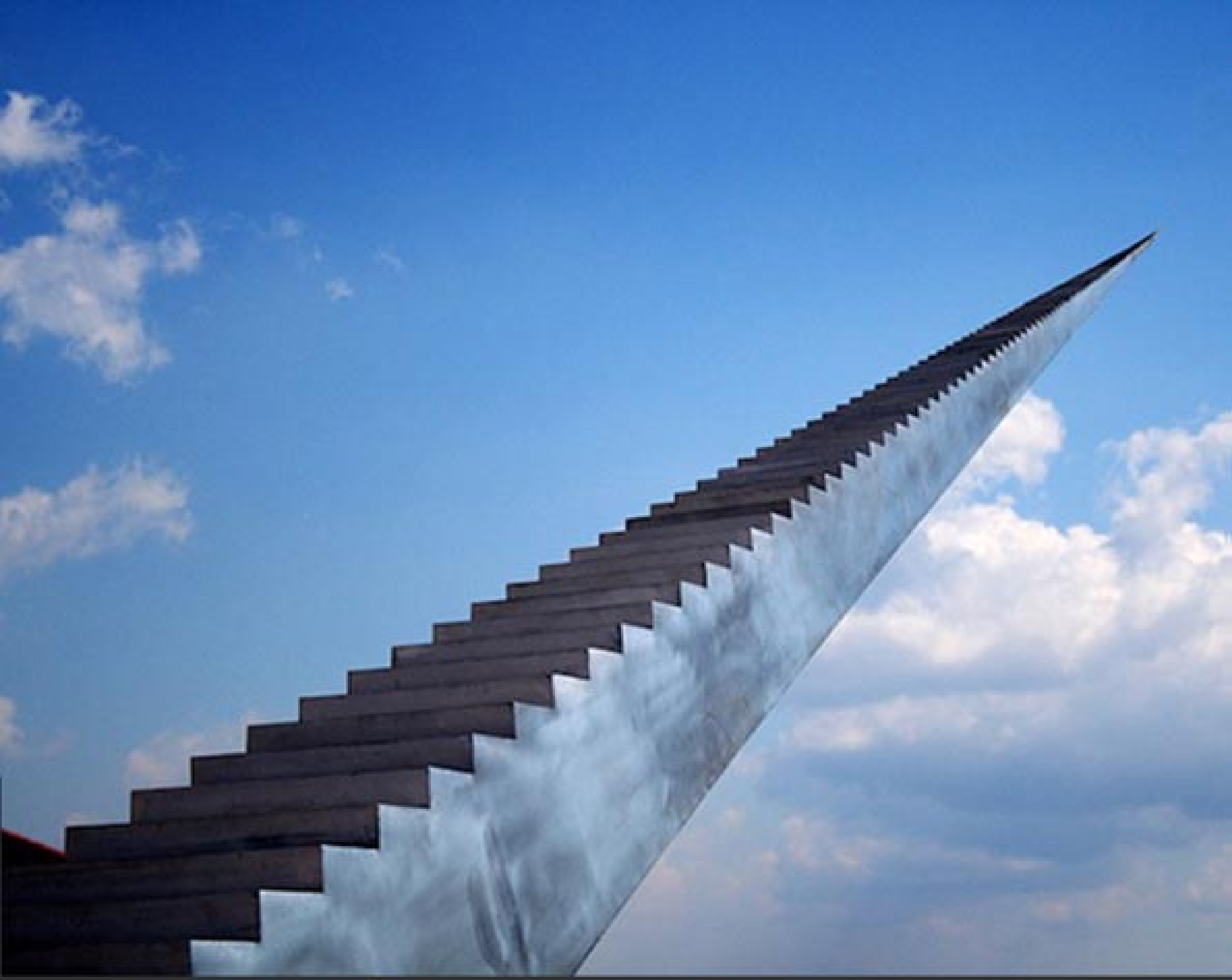


magestore
Magento Extensions of Innovative Functions

MAGENTO MADE EASY

1

Comprehensive Guide to Magento Setup And Development

MAGENTO MADE EASY

Comprehensive Guide to Magento Setup and Development

(Volume 1)

By Magestore

Hanoi, April 2014

TABLE OF CONTENTS

INTRODUCTION

Magento is an open source e-commerce platform for online merchants. Be a Magento user, you are flexible in setting up and developing website, as well as take control of the look, content and functionality of your store. Moreover, Magento team supports you with many marketing, search engine optimization and catalog-management tools.

Now Magento has three editions: Magento Community, Magento Go and Magento Enterprise. Magento Community edition is a free downloadable version. You can modify it as you need and gain supported from Magento Discussion Forum. Magento Go edition is a paid hosted e-commerce solution for small businesses. This version is similar to Magento Community, but you can get some supports from Magento team and automatic updates. Magento Enterprise edition is a paid version and is supported fully by Magento team.

Magento Made Easy: Comprehensive Guide to Magento Setup and Development, as its name, is instructions on how to build up a Magento site for both beginners and developers. This e-book is the volume 1 for beginners and includes two parts: Magento Overview and Module Development.

- **Magento Overview**: The first part is an introduction to Magento (architecture, folder and data structure, configuration and naming). It provides you with a basic background of Magento to prepare for the next part.
- **Module Development**: This part has 14 lessons and focuses on many essential topics in Magento. They are module, menu, grid, form, layout and template, JavaScript and CSS, email, events and class override.

This e-book can be used with other Magento books to optimize an e-commerce website, especially with another e-book by Magestore ***"How to pass Magento Certification Exam in 30 Days"***.

Magento Made Easy: Comprehensive Guide to Magento Setup and Development is a necessary book for both storeowners and developers who want to develop a website on their own. So turn the page and begin now!

April 2014

Magestore Team

ABOUT THE AUTHORS

Magestore is a top Magento Extension Provider founded in January 2010. We have operated and developed with the slogan "Magento Extensions of Innovative Functions" for four years. Now, we have more than 40 extensions and a lot of them are best-sellers for years such as *Affiliate Plus*, *Reward Points*, *Inventory Management*, *One Step Checkout*, *Gift Card*, *Simi POS and SimiCart*.

At Magestore, we always do our best to build the most useful extensions to meet the customers' needs. We divide our extensions into core and additional plug-ins, so you can choose and pay for what is useful to you only. Moreover, our top priority is customers' satisfaction. We offer lifetime support and update for free to guarantee this. Our Support team with experienced and dedicated developers will never let any customer down with us after sales service.

Customers' satisfactory experience is always the encouragement to us to work and improve our products and services. As a result, the e-book ***"Magento Made Easy: Comprehensive Guide to Magento Setup and Development" Volume 1*** was released. It is the work of Magestore technical team members. They are David Nguyen, Alex Nguyen, Michael Nguyen, Blanka Pham, Johnny Giap, Marko Pham and Zackie Duong. Then, the Marketing team translated and edited this e-book.

We hope this e-book is useful for you and thank you for spending your time on it. If you have any comments or recommendations, please contact us:

Visit our website: **http://www.magestore.com/magento-extensions.html**

Follow us on Twitter: **https://twitter.com/magestore**

Friend us on Facebook: **https://www.facebook.com/magestore**

Subcribe to our blog: /**http://blog.magestore.com/**

Your feedback will help us make this e-book better for Magento users.

Read and share Magento Open Course with your friends and colleagues!

PART 01: MAGENTO OVERVIEW

Lesson 01: Setting up a Magento Website

I. OVERVIEW

Developing themes for Magento might be something many developers would like to look into. However, Magento installation can be a bit intimidating especially for designers who just want to get it working and they can start developing themes and other pretty things.

This lesson is a step-by-step procedure for Magento installation. It is written for XAMPP and Windows.

1. Time: 2 hours

2. Content

- Webserver setup and PHP configuration
- Magento installation with sample data

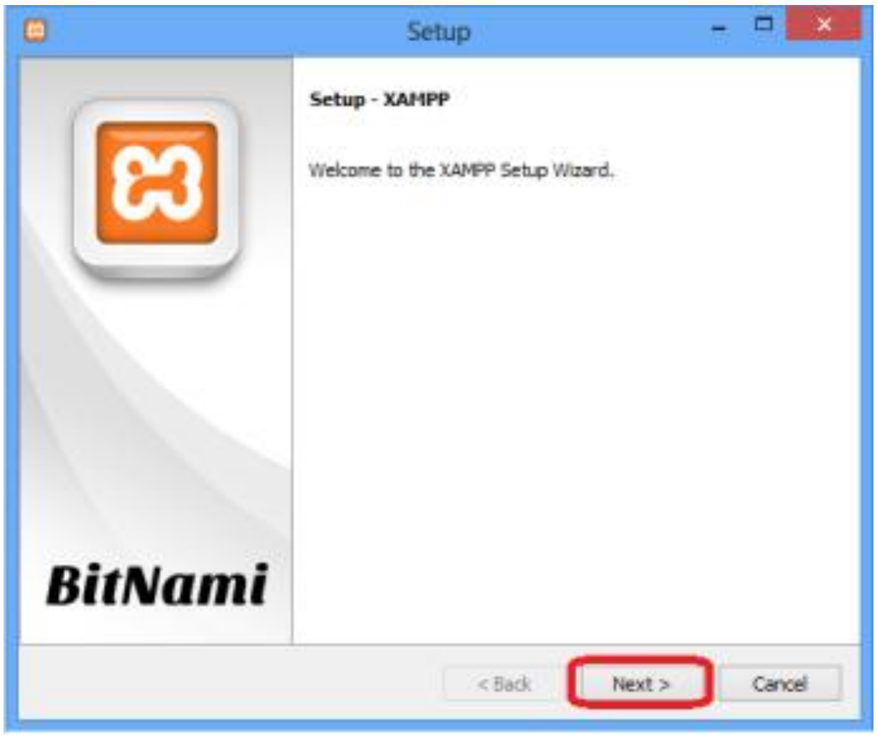
II. CONTENT

1. Webserver setup and PHP configuration

1.1. Set up webserver

Make sure you have the latest version of XAMPP installed. You can get XAMPP on the Apache Friends website http://www.apachefriends.org/en/xampp.html (This document is written based on XAMPP version 1.8.2).

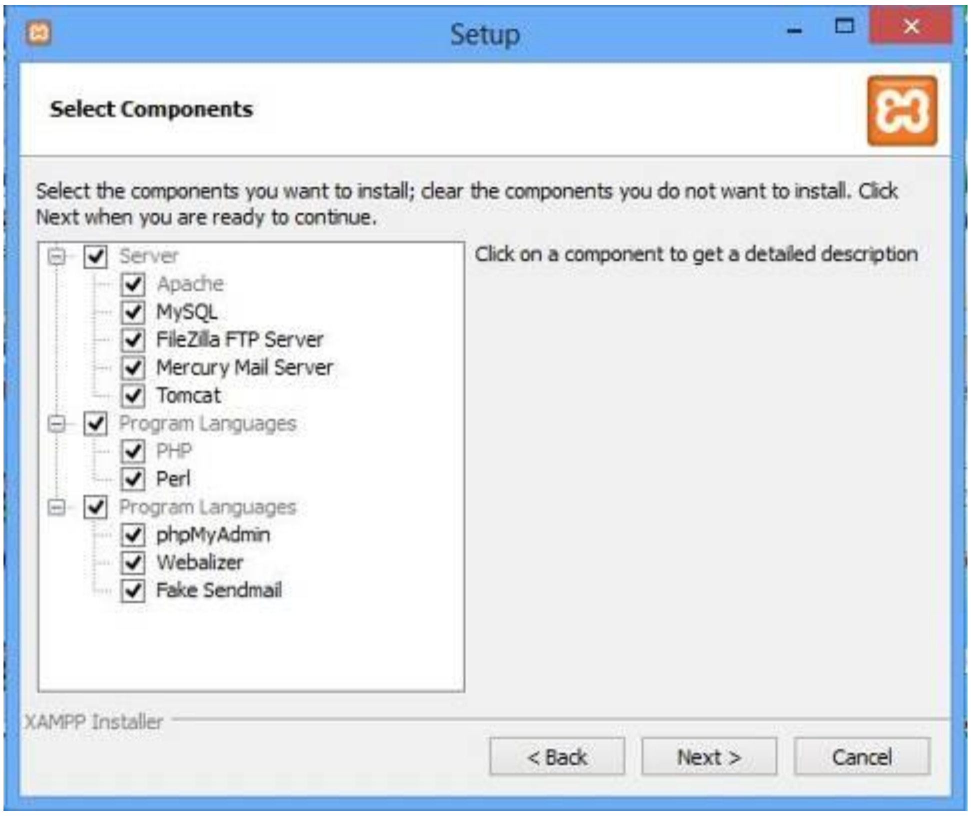
After downloading XAMPP, click on the file to install it on your computer.



Setup - XAMPP

Welcome to the XAMPP Setup Wizard.

Select Components: Select the same as in the image.



Select Components

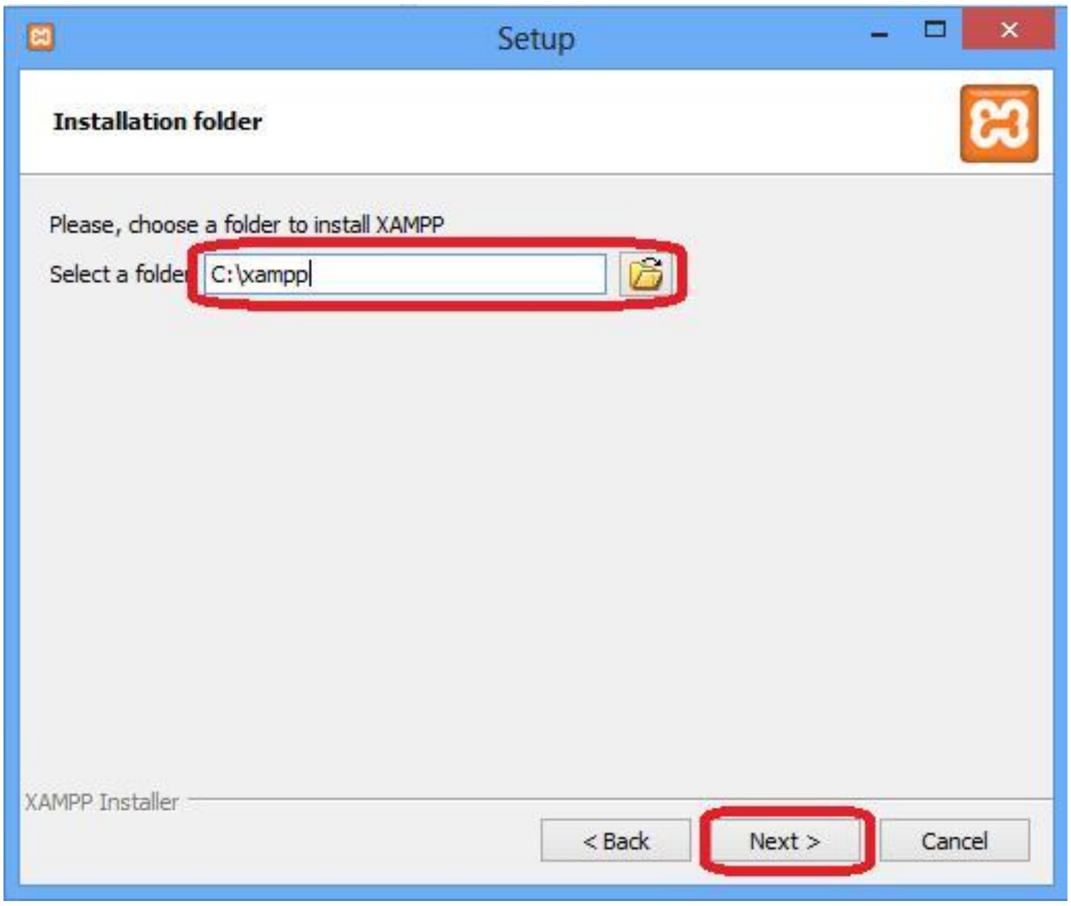
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] Server
  - [✓] Apache
  - [✓] MySQL
  - [✓] FileZilla FTP Server
  - [✓] Mercury Mail Server
  - [✓] Tomcat
- [✓] Program Languages
  - [✓] PHP
  - [✓] Perl
- [✓] Program Languages
  - [✓] phpMyAdmin
  - [✓] Webalizer
  - [✓] Fake Sendmail

Click on a component to get a detailed description

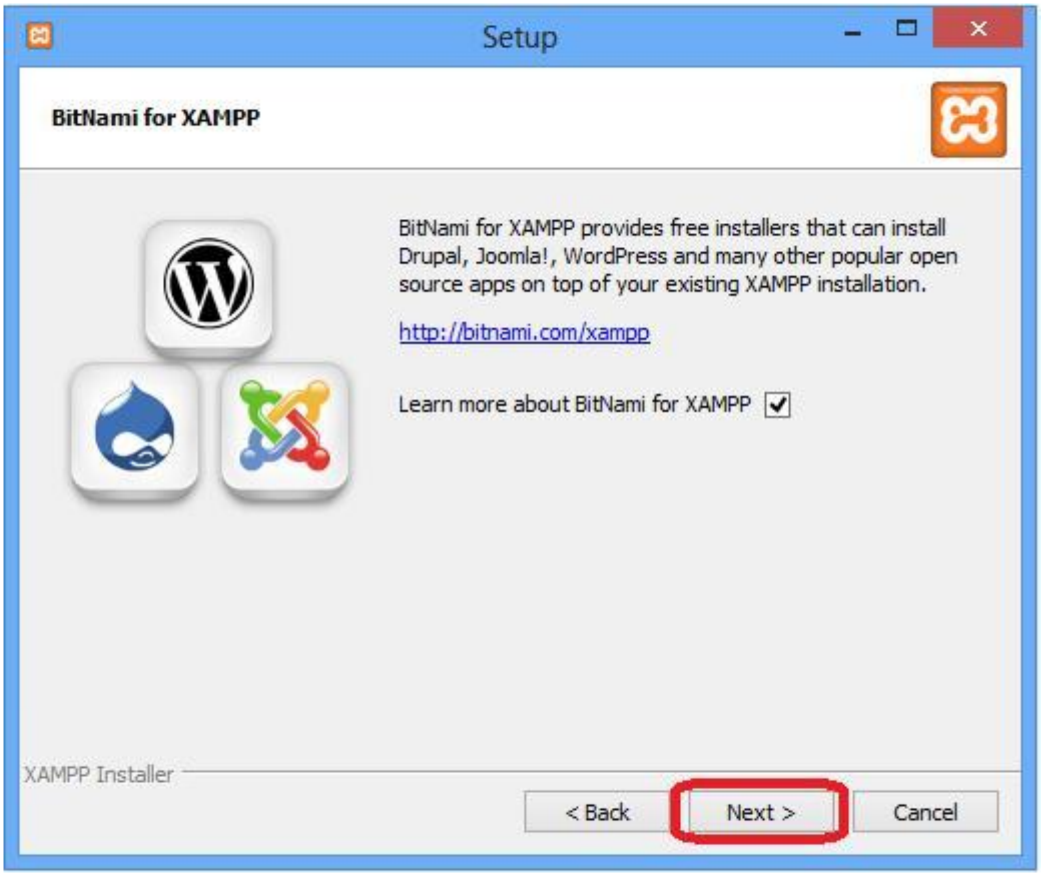
Choose Installation folder: XAMPP default location is C:\xampp. If you need to change the destination, click on the **Browse** button to change your destination for XAMPP program.

Then, click on the **Next** button to go to the next step.



Please, choose a folder to install XAMPP

Select a folder C:\xampp

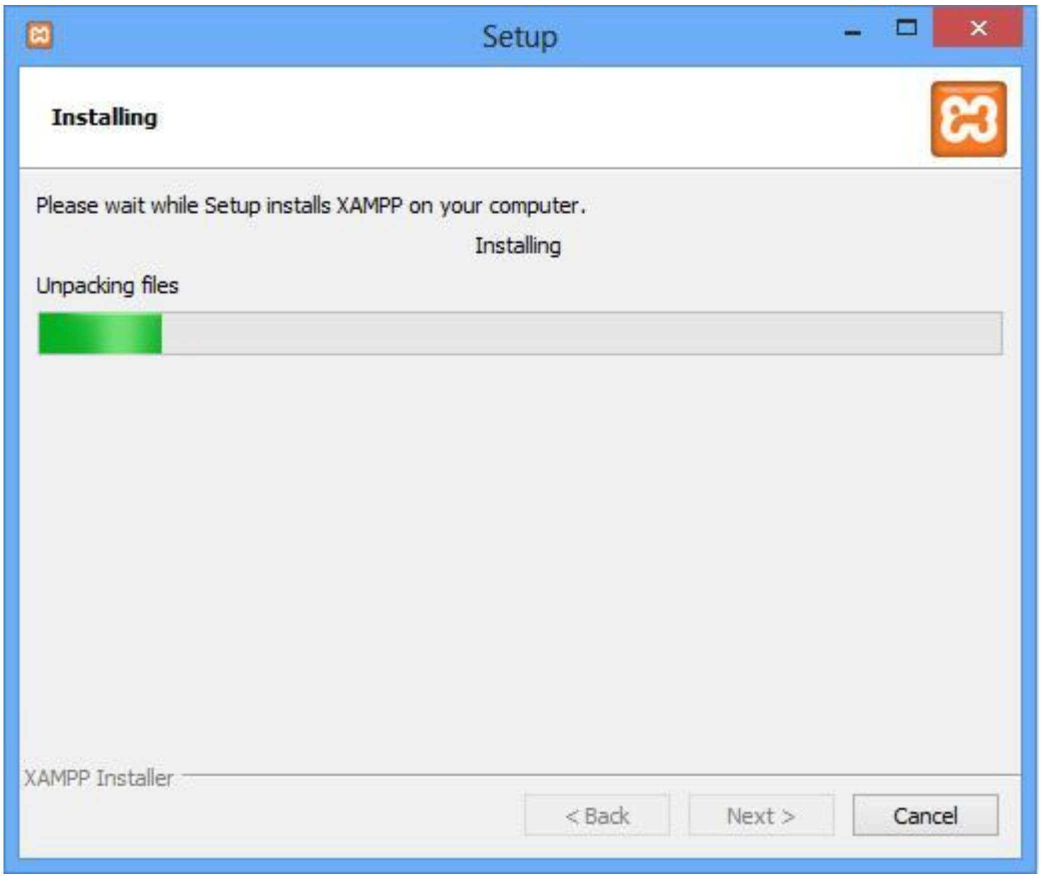


BitNami for XAMPP provides free installers that can install Drupal, Joomla!, WordPress and many other popular open source apps on top of your existing XAMPP installation.

http://bitnami.com/xampp

Learn more about BitNami for XAMPP ✓

Start installing XAMPP



Please wait while Setup installs XAMPP on your computer.

Installing

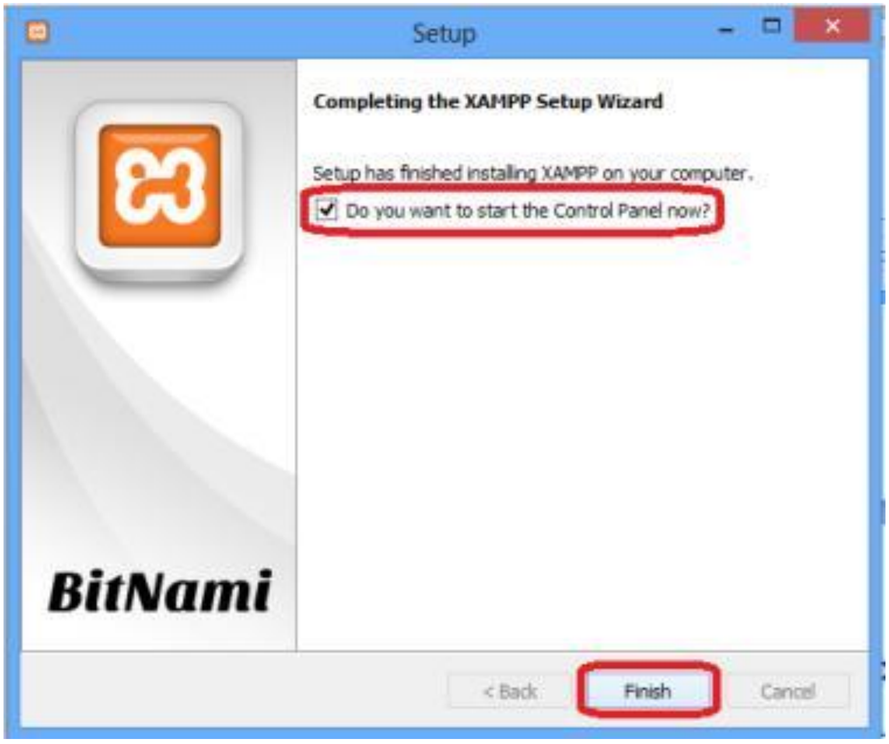
Unpacking files

XAMPP Installer

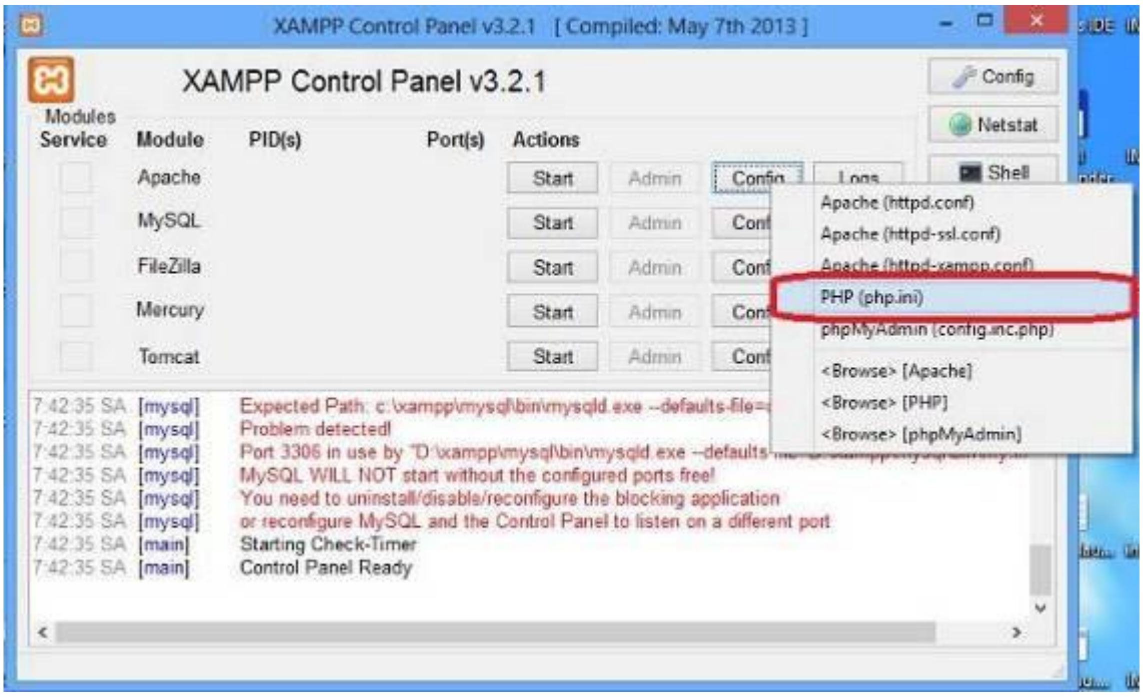
Complete the XAMPP setup.

Click on the **Finish** button to end this setup and XAMPP prompt.

If you want to start the program right away, tick the box when the setup asks you: "Do you want to start the Control Panel now?"


Setup
Completing the XAMPP Setup Wizard
Setup has finished installing XAMPP on your computer.
☑ Do you want to start the Control Panel now?
BitNami
< Back
Finish
Cancel

1.2. Configure PHP



```
7:42:35 SA  [mysql]    Expected Path: c:\xampp\mysql\bin\mysqld.exe --defaults-file=
7:42:35 SA  [mysql]    Problem detected!
7:42:35 SA  [mysql]    Port 3306 in use by "D:\xampp\mysql\bin\mysqld.exe --defaults
7:42:35 SA  [mysql]    MySQL WILL NOT start without the configured ports free!
7:42:35 SA  [mysql]    You need to uninstall/disable/reconfigure the blocking application
7:42:35 SA  [mysql]    or reconfigure MySQL and the Control Panel to listen on a different port
7:42:35 SA  [main]     Starting Check-Timer
7:42:35 SA  [main]     Control Panel Ready
```

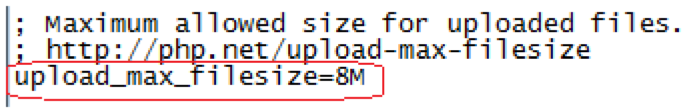
In the XAMPP Control Panel, you can see the row Apache. Click on the **Config** button and click PHP (php.ini) then remove the comment mark ";" in following rows:

```
extension=php_curl.dll
extension=php_mcrypt.dll
extension=php_pdo_sqlite.dll
```

extension=php_pdo_mysql.dll

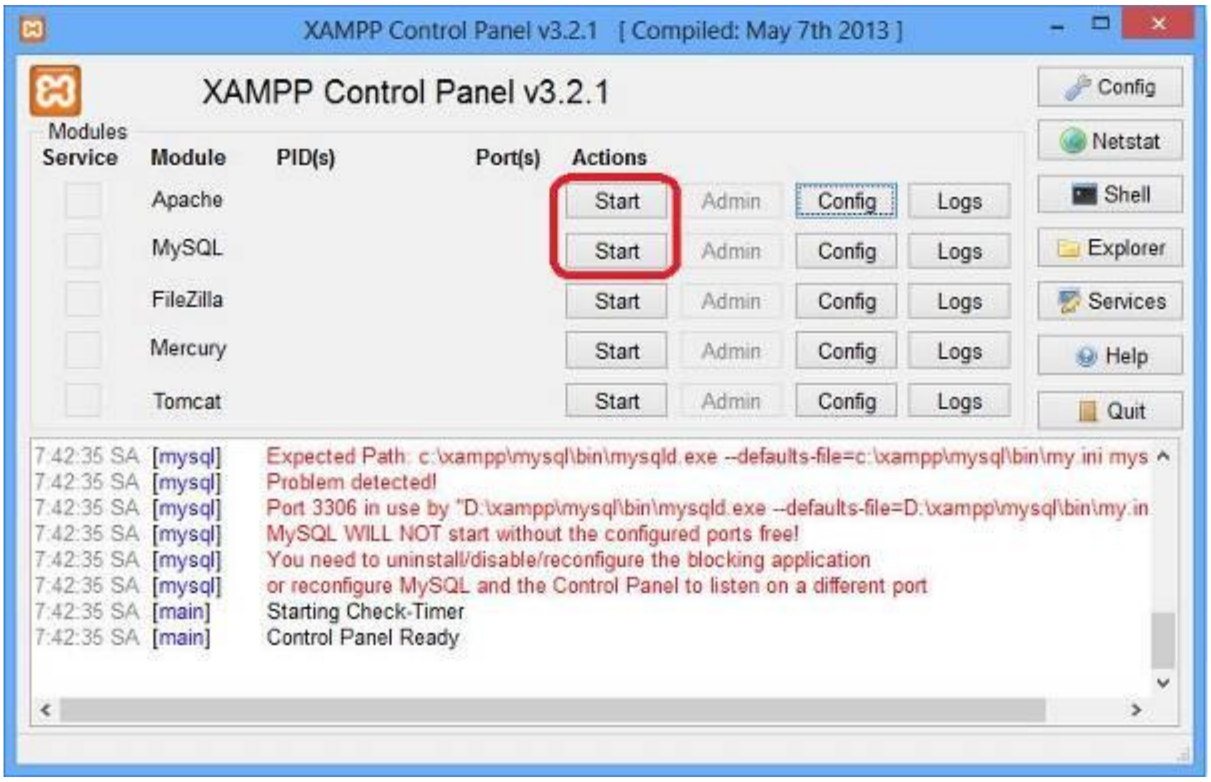
extension=php_soap.dll

Search for “upload_max_filesize” and change the filesize limit from 2M to 8M like this:



```
; Maximum allowed size for uploaded files.
; http://php.net/upload-max-filesize
upload_max_filesize=8M
```

After that, click on the **Start** button on two rows - Apache and MySQL - to start them:



```
7:42:35 SA  [mysql]   Expected Path: c:\xampp\mysql\bin\mysqld.exe --defaults-file=c:\xampp\mysql\bin\my.ini mys
7:42:35 SA  [mysql]   Problem detected!
7:42:35 SA  [mysql]   Port 3306 in use by "D:\xampp\mysql\bin\mysqld.exe --defaults-file=D:\xampp\mysql\bin\my.in
7:42:35 SA  [mysql]   MySQL WILL NOT start without the configured ports free!
7:42:35 SA  [mysql]   You need to uninstall/disable/reconfigure the blocking application
7:42:35 SA  [mysql]   or reconfigure MySQL and the Control Panel to listen on a different port
7:42:35 SA  [main]    Starting Check-Timer
7:42:35 SA  [main]    Control Panel Ready
```

1.3. Configure hosts file

Open the file C:\Windows\System32\drivers\etc\hosts.

Add the following code to end of the file:

127.0.0.1 www.localhost.com

2. Magento installation with sample data

2.1. Download Magento and sample data

Download the full latest released archive file from http://magento.com/. In this document, we use Magento Community Edition v.1.7.0.2 and sample data archive v.1.6.1.0.

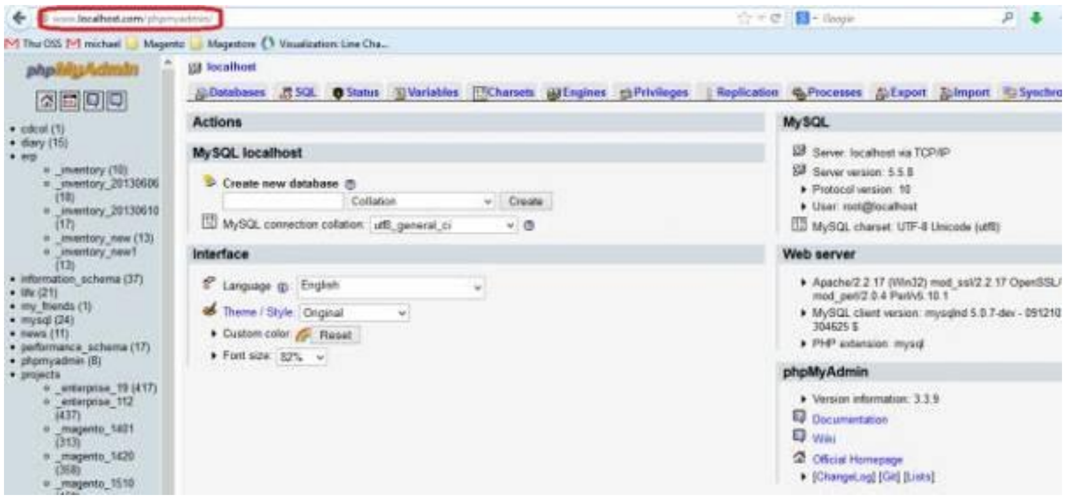
The archive file is available in .zip, .tar.gz, and .tar.bz2 formats for downloading (a zipped archive contains exactly the same files and is provided in different formats for your convenience).

On your computer, extract Magento from the downloaded archive. Magento has a folder named Magento containing all of the required Magento files. The sample data package has a data file called “magento_sample_data_for_1.6.1.0.sql” and a folder called “media”.

2.2. Import the Magento sample data into an empty store database

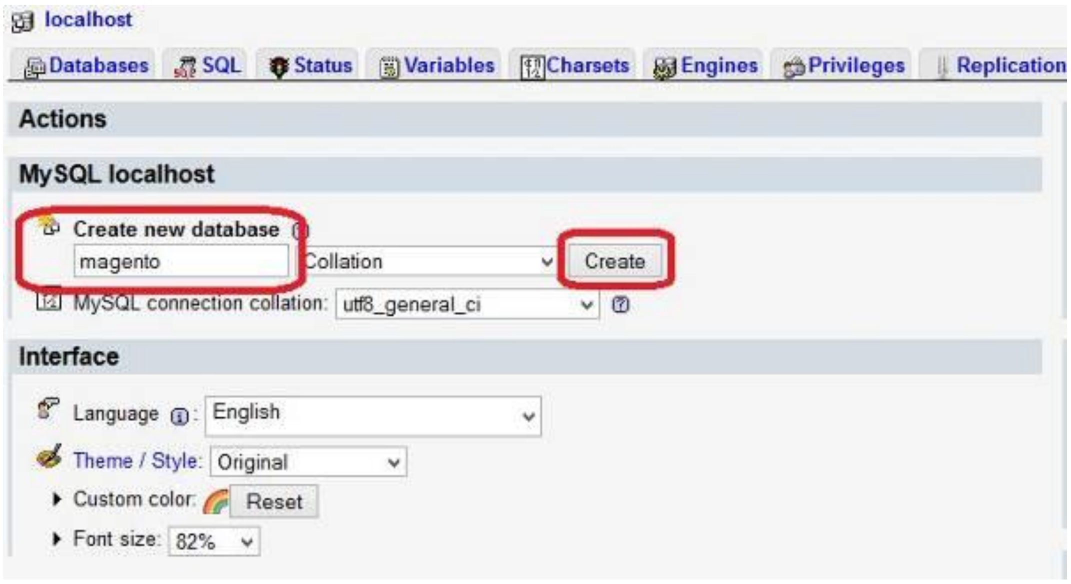
Create a new empty database for Magento.

Using a browser, enter the URL http://www.localhost.com/phpmyadmin/ to create a new empty database by using phpMyAdmin:



MySQL connection collation: utf8_general_ci
Language: English
Theme / Style: Original
Font size: 82%
Server: localhost via TCP/IP
Server version: 5.5.8
Protocol version: 10
User: root@localhost
MySQL charset: UTF-8 Unicode (utf8)
Version information: 3.3.9

Create an empty database named “Magento”:

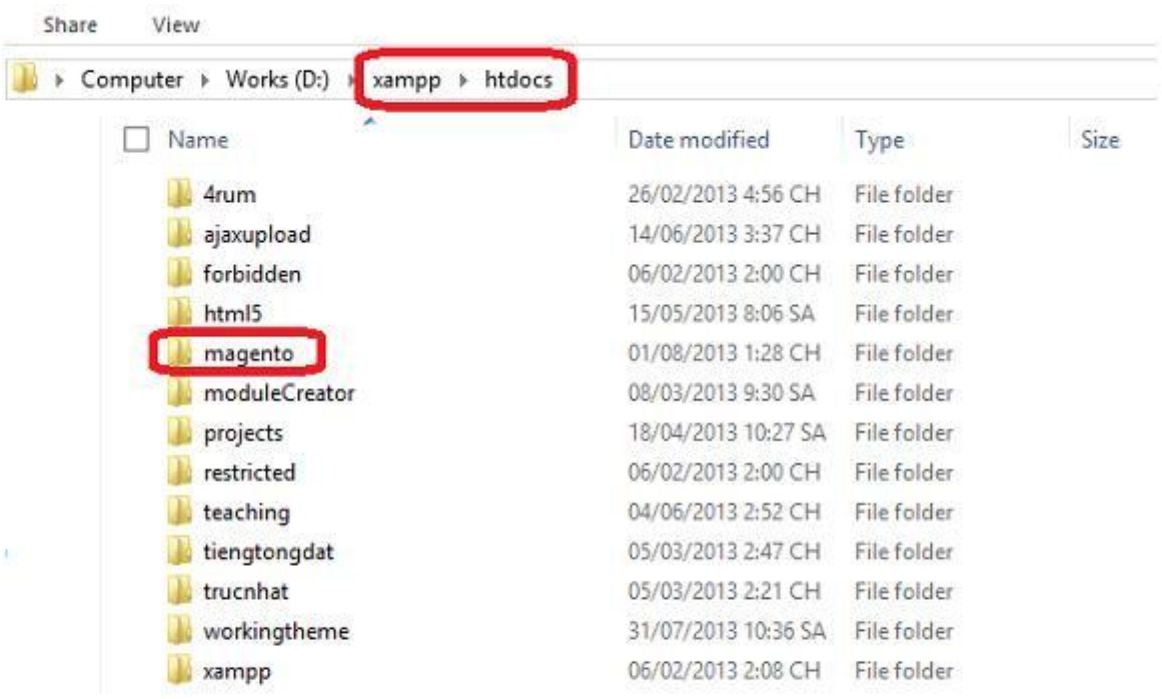


Create new database: magento
MySQL connection collation: utf8_general_ci
Language: English
Theme / Style: Original
Font size: 82%

Import the .sql file of the sample data (magento_sample_data_for_1.6.1.0.sql) into your empty Magento database. Use the import function of your database management tool to import the data in the .sql file into your empty database.

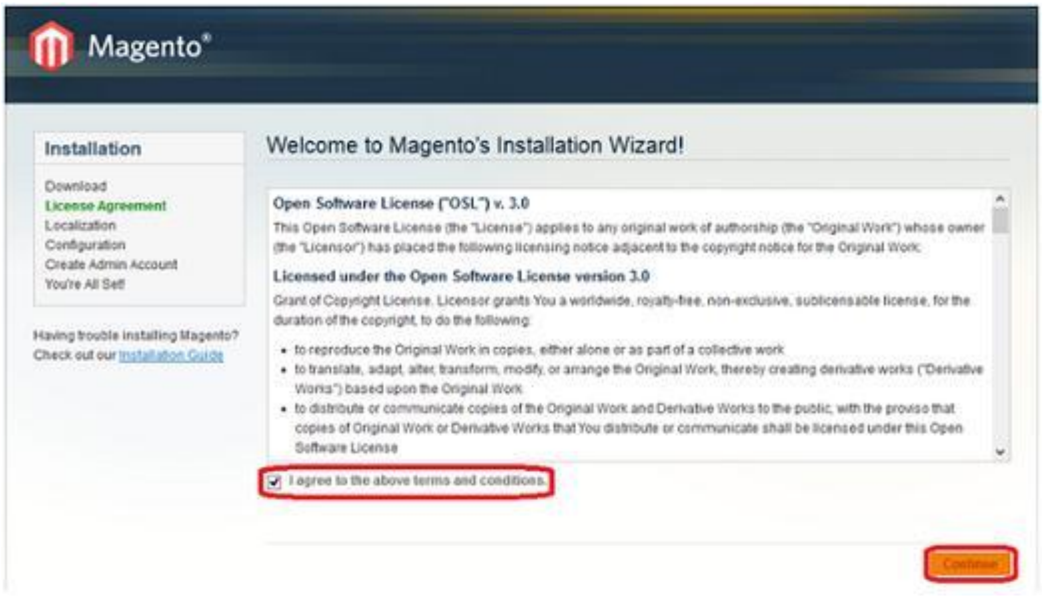
2.3. Install Magento

First, restart your Apache and MySQL servers. If you are editing them while they are on, stop and restart them. Move or copy the Magento folder after extracting Magento from the downloaded archive to xampp\htdocs\:



Name	Date modified	Type	Size
4rum	26/02/2013 4:56 CH	File folder	
ajaxupload	14/06/2013 3:37 CH	File folder	
forbidden	06/02/2013 2:00 CH	File folder	
html5	15/05/2013 8:06 SA	File folder	
magento	01/08/2013 1:28 CH	File folder	
moduleCreator	08/03/2013 9:30 SA	File folder	
projects	18/04/2013 10:27 SA	File folder	
restricted	06/02/2013 2:00 CH	File folder	
teaching	04/06/2013 2:52 CH	File folder	
tiengtongdat	05/03/2013 2:47 CH	File folder	
trucnhat	05/03/2013 2:21 CH	File folder	
workingtheme	31/07/2013 10:36 SA	File folder	
xampp	06/02/2013 2:08 CH	File folder	

Using a browser, enter the URL http://www.localhost.com/magento/ to start installing Magento:



Welcome to Magento's Installation Wizard!

Open Software License ("OSL") v. 3.0

This Open Software License (the "License") applies to any original work of authorship (the "Original Work") whose owner (the "Licensor") has placed the following licensing notice adjacent to the copyright notice for the Original Work:

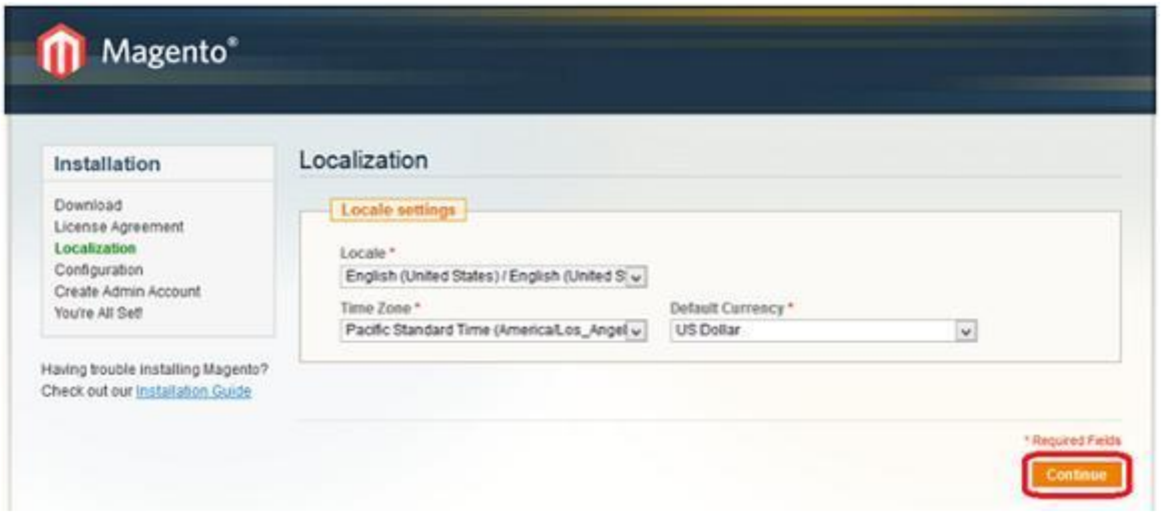
Licensed under the Open Software License version 3.0

Grant of Copyright License. Licensor grants You a worldwide, royalty-free, non-exclusive, sublicensable license, for the duration of the copyright, to do the following:

- to reproduce the Original Work in copies, either alone or as part of a collective work
- to translate, adapt, alter, transform, modify, or arrange the Original Work, thereby creating derivative works ("Derivative Works") based upon the Original Work
- to distribute or communicate copies of the Original Work and Derivative Works to the public, with the proviso that copies of Original Work or Derivative Works that You distribute or communicate shall be licensed under this Open Software License

☑ I agree to the above terms and conditions.

Tick the checkbox **"I agree to the above terms and conditions"** and click on the **Continue** button to continue. You might want to change the Time Zone, Locale and Currency before continuing:



Localization

Locale settings

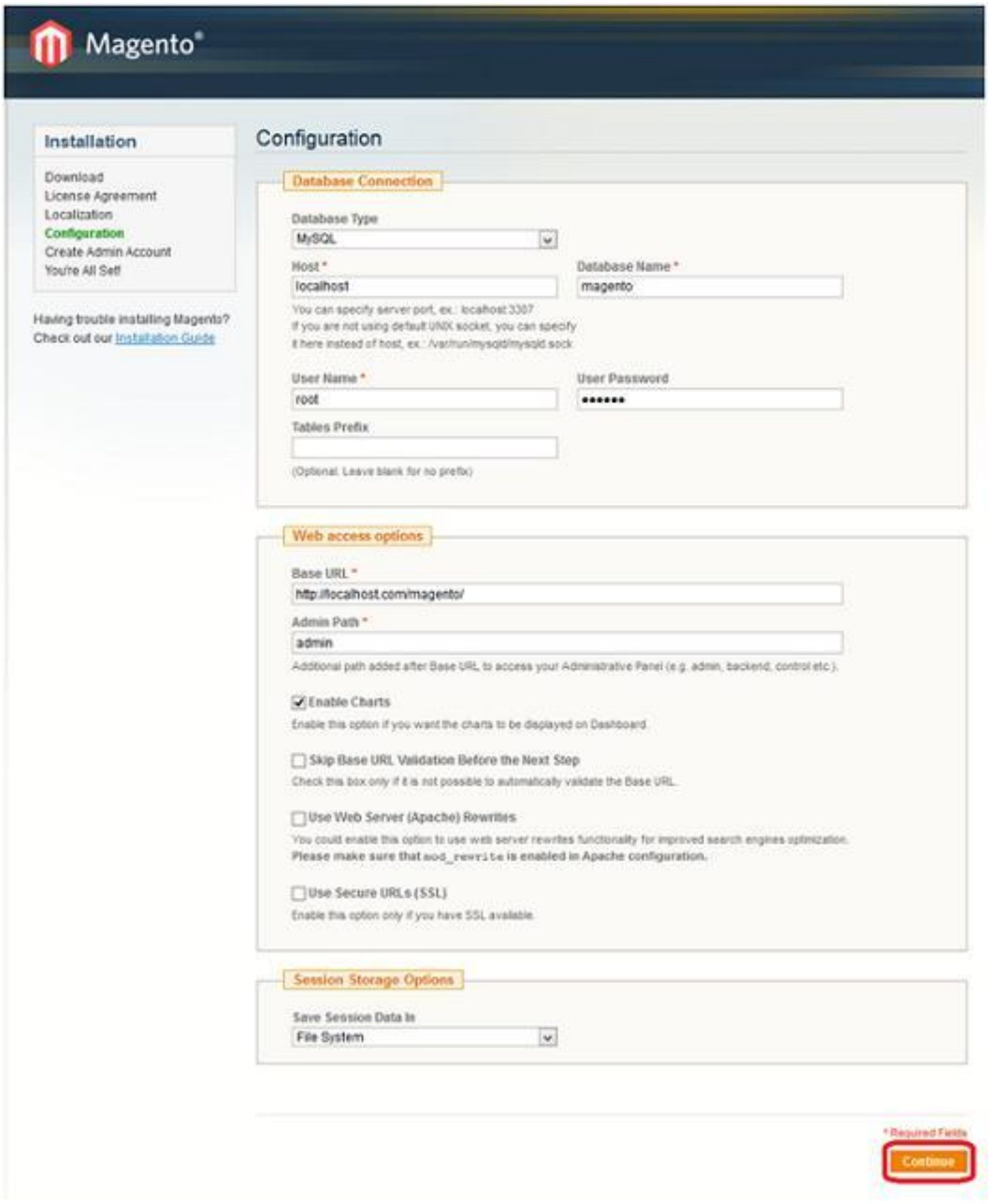
Locale *: English (United States) / English (United S

Time Zone *: Pacific Standard Time (America/Los_Angel

Default Currency *: US Dollar

* Required Fields

Next, fill in the Database Connection form: Host, Database Name, User Name and User Password:


Installation

Download
License Agreement
Localization
Configuration
Create Admin Account
You're All Set!

Having trouble installing Magento?
Check out our Installation Guide

Configuration

Database Connection

Database Type: MySQL

Host *: localhost
You can specify server port, ex.: localhost:3307
If you are not using default UNIX socket, you can specify it here instead of host, ex.: /var/run/mysqld/mysqld.sock

Database Name *: magento

User Name *: root

User Password: ••••••

Tables Prefix:
(Optional. Leave blank for no prefix)

Web access options

Base URL *: http://localhost.com/magento/

Admin Path *: admin
Additional path added after Base URL to access your Administrative Panel (e.g. admin, backend, control etc.).

[✓] Enable Charts
Enable this option if you want the charts to be displayed on Dashboard.

[ ] Skip Base URL Validation Before the Next Step
Check this box only if it is not possible to automatically validate the Base URL.

[ ] Use Web Server (Apache) Rewrites
You could enable this option to use web server rewrites functionality for improved search engines optimization.
Please make sure that `mod_rewrite` is enabled in Apache configuration.

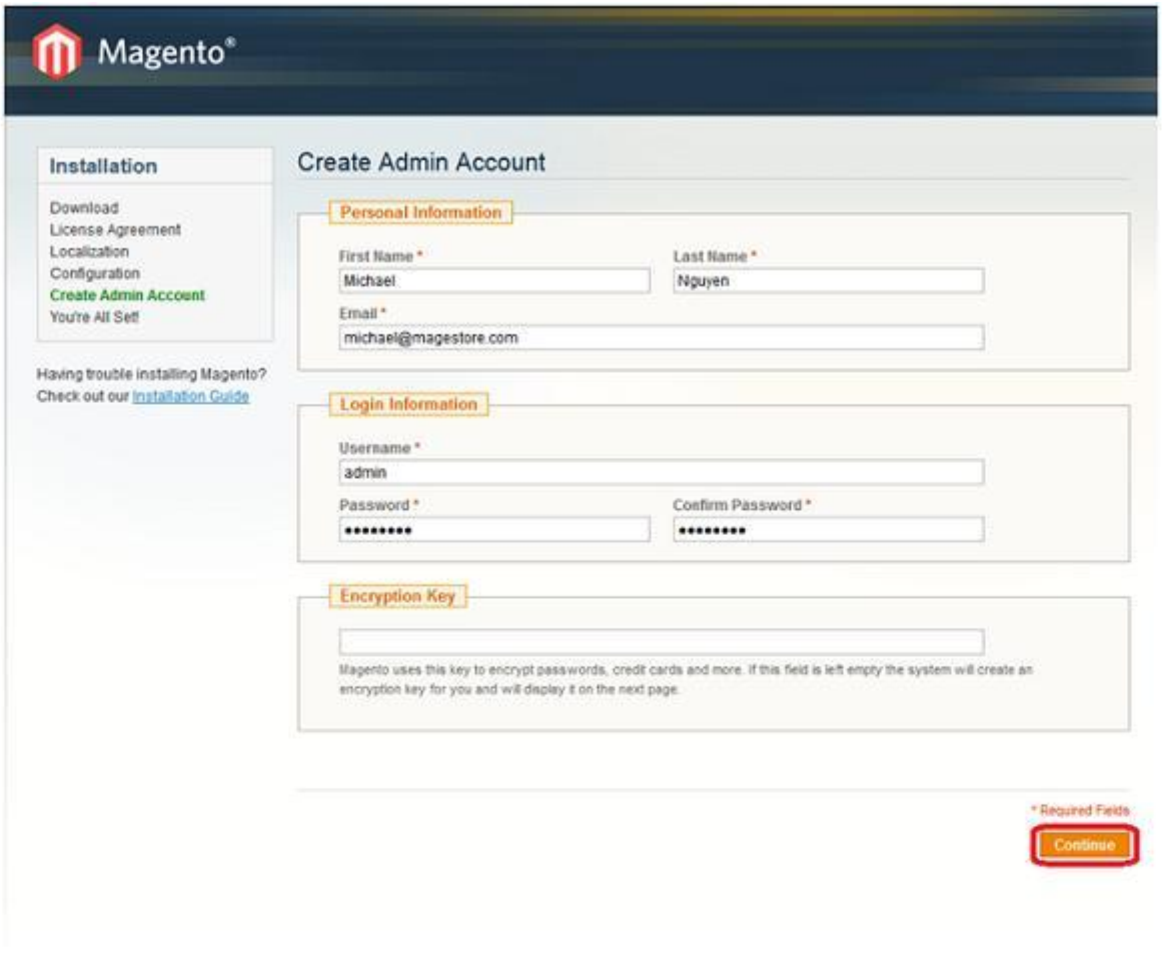
[ ] Use Secure URLs (SSL)
Enable this option only if you have SSL available.

Session Storage Options

Save Session Data In: File System

* Required Fields

Then, fill in the Personal Information: First Name, Last Name and Email and the Login Information to admin use in backend: Username, Password and Confirm Password



Having trouble installing Magento?
Check out our Installation Guide

Create Admin Account

Personal Information

First Name *	Last Name *
Michael	Nguyen
Email *	
michael@magestore.com	

Login Information

Username *	
admin	
Password *	Confirm Password *
••••••••	••••••••

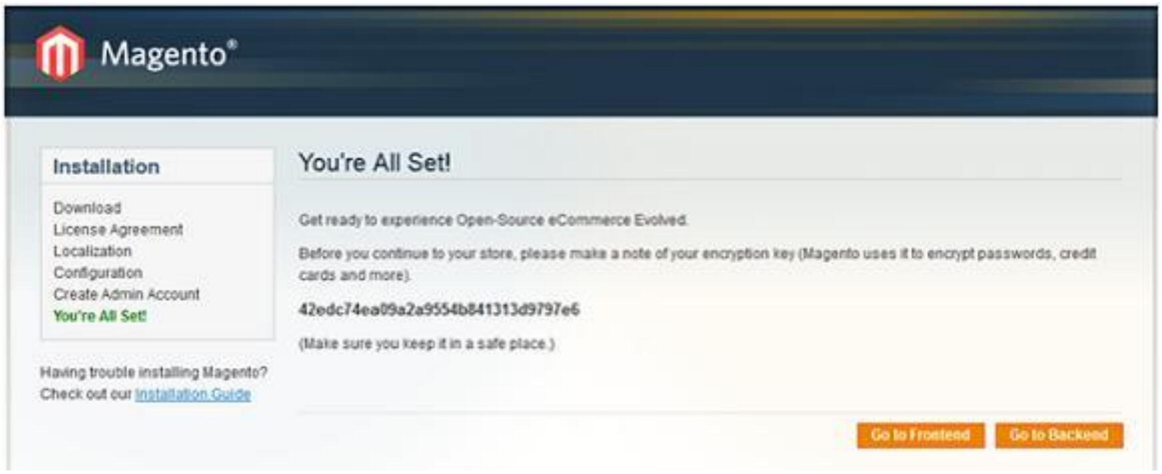
Encryption Key

Magento uses this key to encrypt passwords, credit cards and more. If this field is left empty the system will create an encryption key for you and will display it on the next page.

* Required Fields

You do not need to worry about what to fill in the Encryption Key field. Magento will generate a key for you on the next page. It is highly recommended that you save that key down somewhere so that you will not forget it. Once you have finished filling out the form, click on the "Continue" button and write down your encryption key.

After you get your encryption key locked away in a fireproof safe, you can choose to go to Frontend or Backend of your new Magento website.



Having trouble installing Magento?
Check out our Installation Guide

You're All Set!

Get ready to experience Open-Source eCommerce Evolved.

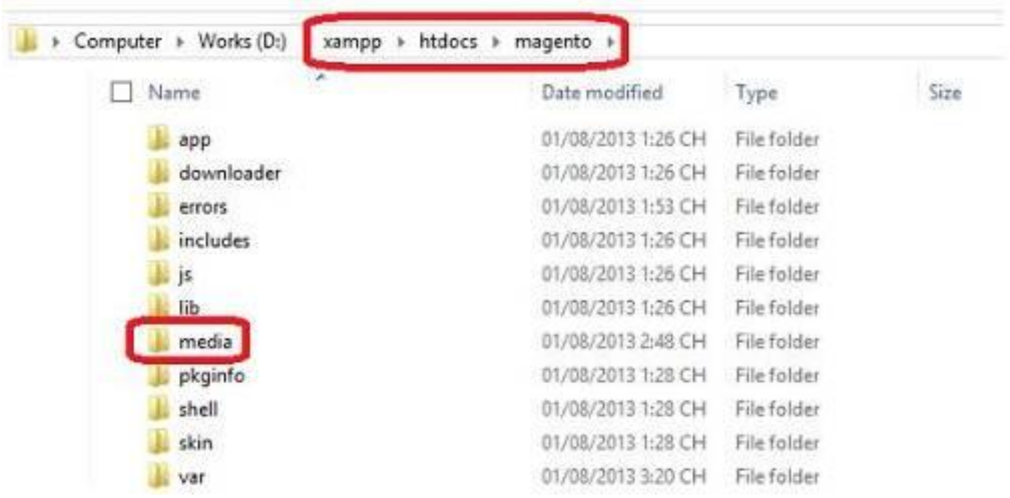
Before you continue to your store, please make a note of your encryption key (Magento uses it to encrypt passwords, credit cards and more).

42edc74ea09a2a9554b841313d9797e6

(Make sure you keep it in a safe place.)

2.4. Copy media to source

Copy media folder after extracting the Sample data from the downloaded archive to xampp\htdocs\magento\

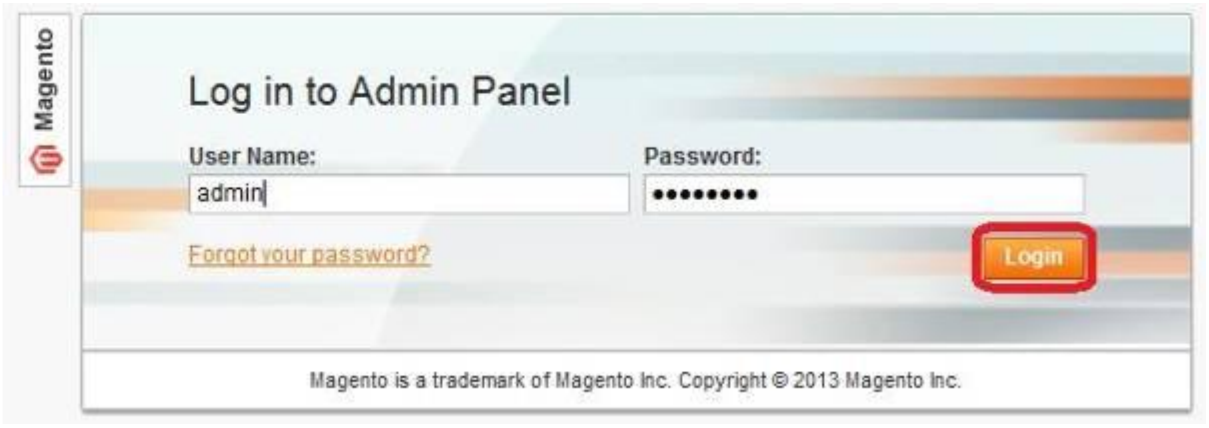


```
Computer ▸ Works (D:) ▸ xampp ▸ htdocs ▸ magento ▸

Name            Date modified        Type          Size
app             01/08/2013 1:26 CH   File folder
downloader      01/08/2013 1:26 CH   File folder
errors          01/08/2013 1:53 CH   File folder
includes        01/08/2013 1:26 CH   File folder
js              01/08/2013 1:26 CH   File folder
lib             01/08/2013 1:26 CH   File folder
media           01/08/2013 2:48 CH   File folder
pkginfo         01/08/2013 1:28 CH   File folder
shell           01/08/2013 1:28 CH   File folder
skin            01/08/2013 1:28 CH   File folder
var             01/08/2013 3:20 CH   File folder
```

2.5. Refresh cache and re-index data

Click on the **"Go to Backend"** button in section 2.3 or type www.localhost.com/magento/index.php/admin/ to your web browser.

You can see the admin login page. Fill in the **Personal Information** (First Name, Last Name and Email) and **Login Information** to admin use in backend (Username, Password and Confirm Password).



Log in to Admin Panel

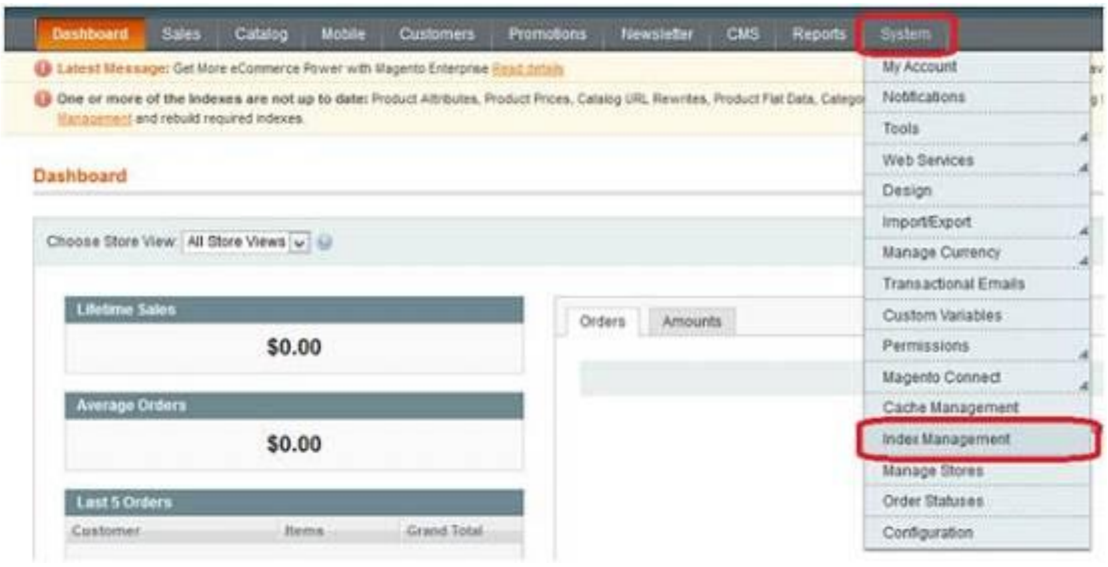
User Name: admin

Password: ••••••••

Magento is a trademark of Magento Inc. Copyright © 2013 Magento Inc.

Re-index data:

On the admin page, go to the System menu and click Index Management.



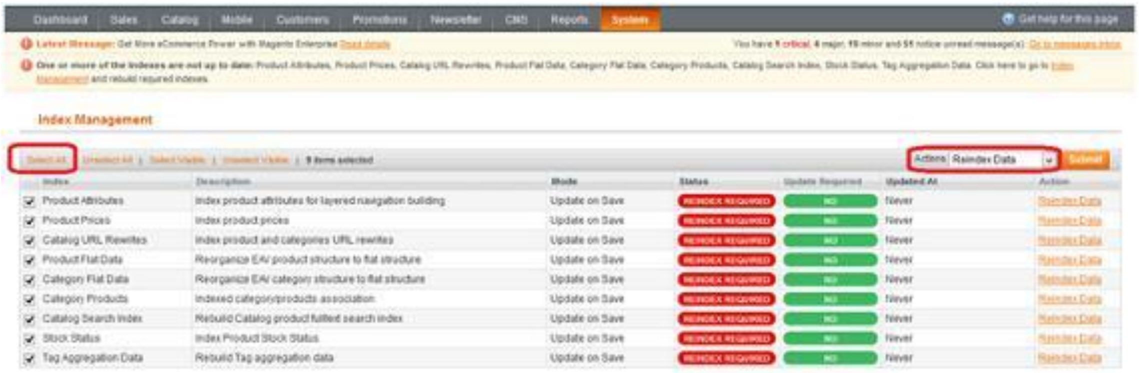
Latest Message: Get More eCommerce Power with Magento Enterprise Read details

One or more of the Indexes are not up to date: Product Attributes, Product Prices, Catalog URL Rewrites, Product Flat Data, Catego

Management and rebuild required indexes.

You can see the Index Management page.

First, click Select All. In Action field, select Reindex Data before clicking on the Submit button:



Latest Message: Get More eCommerce Power with Magento Enterprise Read details

You have 1 critical, 4 major, 19 minor and 51 notice unread message(s). Go to messages inbox

One or more of the Indexes are not up to date: Product Attributes, Product Prices, Catalog URL Rewrites, Product Flat Data, Category Flat Data, Category Products, Catalog Search Index, Stock Status, Tag Aggregation Data. Click here to go to Index Management and rebuild required indexes.

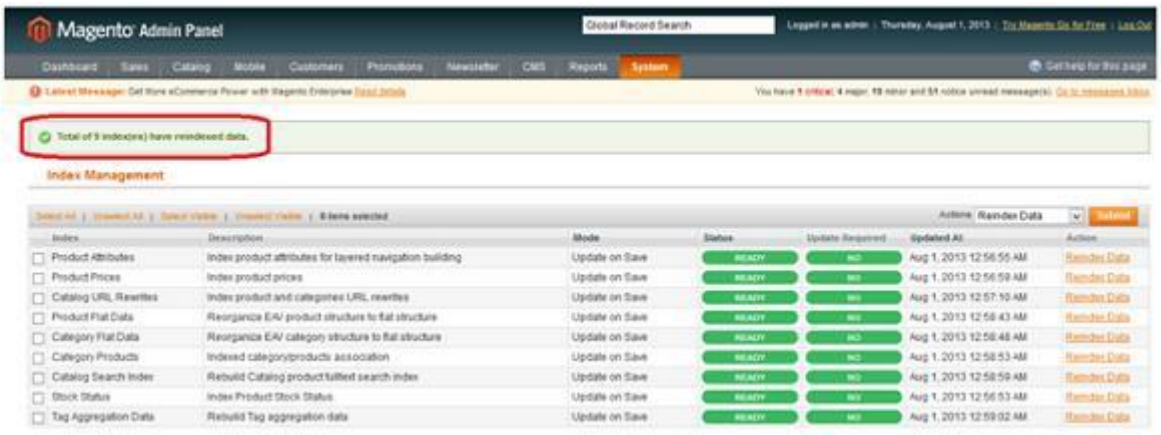
Index Management

Select All | Unselect All | Select Visible | Unselect Visible | 9 items selected

Actions: Reindex Data — Submit

	Index	Description	Mode	Status	Update Required	Updated At	Action
✓	Product Attributes	Index product attributes for layered navigation building	Update on Save	REINDEX REQUIRED	NO	Never	Reindex Data
✓	Product Prices	Index product prices	Update on Save	REINDEX REQUIRED	NO	Never	Reindex Data
✓	Catalog URL Rewrites	Index product and categories URL rewrites	Update on Save	REINDEX REQUIRED	NO	Never	Reindex Data
✓	Product Flat Data	Reorganize EAV product structure to flat structure	Update on Save	REINDEX REQUIRED	NO	Never	Reindex Data
✓	Category Flat Data	Reorganize EAV category structure to flat structure	Update on Save	REINDEX REQUIRED	NO	Never	Reindex Data
✓	Category Products	Indexed category/products association	Update on Save	REINDEX REQUIRED	NO	Never	Reindex Data
✓	Catalog Search Index	Rebuild Catalog product fulltext search index	Update on Save	REINDEX REQUIRED	NO	Never	Reindex Data
✓	Stock Status	Index Product Stock Status	Update on Save	REINDEX REQUIRED	NO	Never	Reindex Data
✓	Tag Aggregation Data	Rebuild Tag aggregation data	Update on Save	REINDEX REQUIRED	NO	Never	Reindex Data

After the system re-indexes data, a success message will be shown as follows:



Magento Admin Panel

Global Record Search

Logged in as admin | Thursday, August 1, 2013 | Try Magento Go for Free | Log Out

Latest Message: Get More eCommerce Power with Magento Enterprise Read details

You have 1 critical, 4 major, 19 minor and 51 notice unread message(s). Go to messages inbox

Total of 9 index(es) have reindexed data.

Index Management

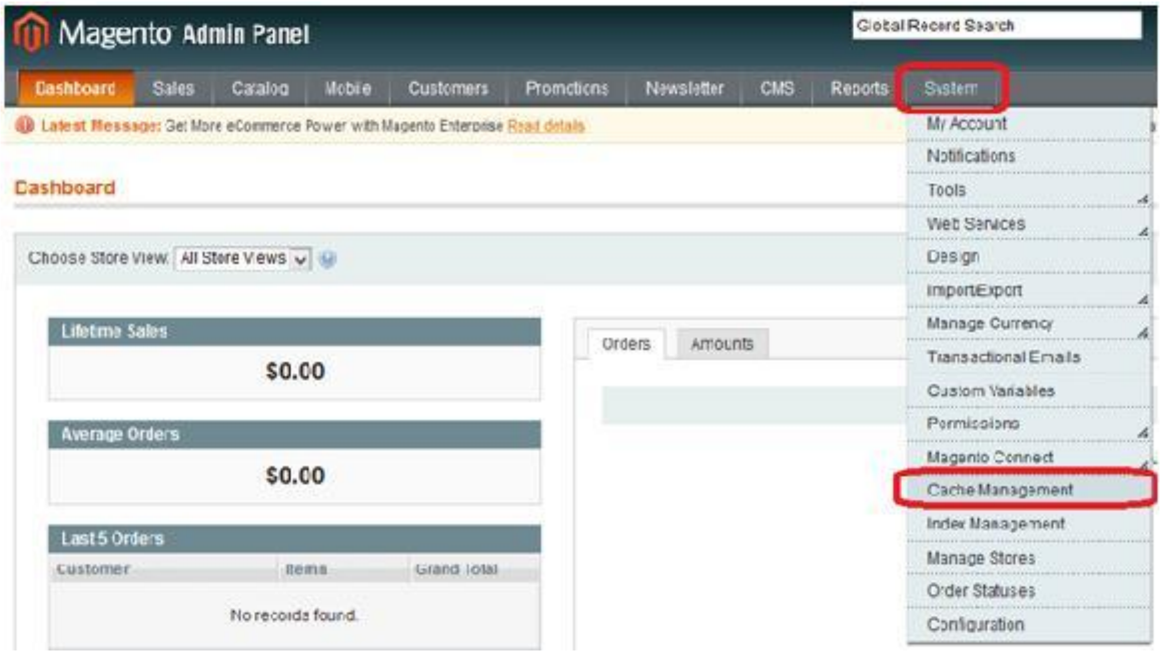
Select All | Unselect All | Select Visible | Unselect Visible | 0 items selected

Actions: Reindex Data — Submit

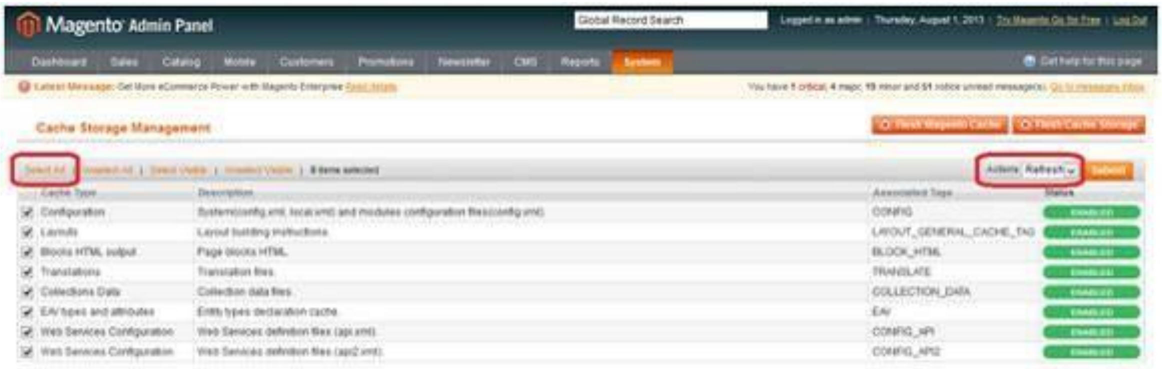
	Index	Description	Mode	Status	Update Required	Updated At	Action
☐	Product Attributes	Index product attributes for layered navigation building	Update on Save	READY	NO	Aug 1, 2013 12:56:55 AM	Reindex Data
☐	Product Prices	Index product prices	Update on Save	READY	NO	Aug 1, 2013 12:56:59 AM	Reindex Data
☐	Catalog URL Rewrites	Index product and categories URL rewrites	Update on Save	READY	NO	Aug 1, 2013 12:57:10 AM	Reindex Data
☐	Product Flat Data	Reorganize EAV product structure to flat structure	Update on Save	READY	NO	Aug 1, 2013 12:58:43 AM	Reindex Data
☐	Category Flat Data	Reorganize EAV category structure to flat structure	Update on Save	READY	NO	Aug 1, 2013 12:58:48 AM	Reindex Data
☐	Category Products	Indexed category/products association	Update on Save	READY	NO	Aug 1, 2013 12:58:53 AM	Reindex Data
☐	Catalog Search Index	Rebuild Catalog product fulltext search index	Update on Save	READY	NO	Aug 1, 2013 12:58:59 AM	Reindex Data
☐	Stock Status	Index Product Stock Status	Update on Save	READY	NO	Aug 1, 2013 12:56:53 AM	Reindex Data
☐	Tag Aggregation Data	Rebuild Tag aggregation data	Update on Save	READY	NO	Aug 1, 2013 12:59:02 AM	Reindex Data

Refresh the cache:

Go to admin page, next see the System menu and click to Cache Management

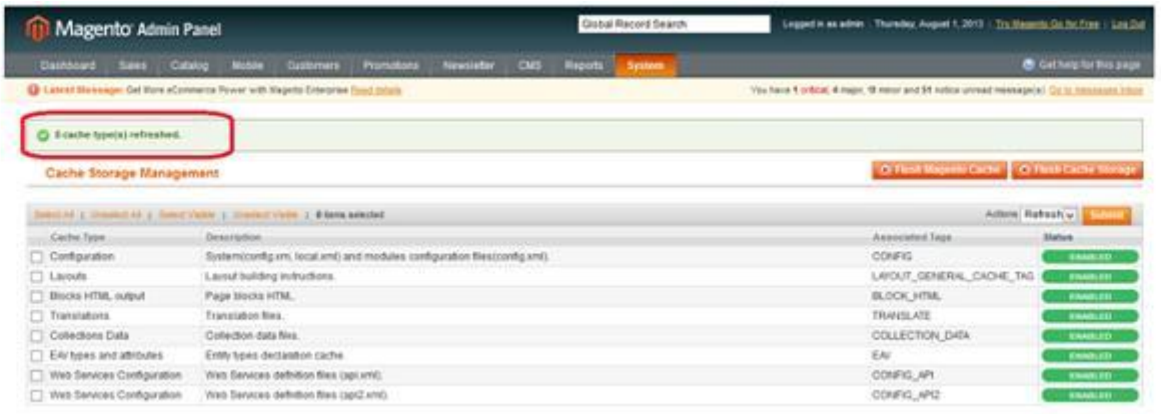


Click **Select All**. In Action field, select Refresh and click on the Submit button:


Magento Admin Panel
Global Record Search
Logged in as admin | Thursday, August 1, 2013 | Try Magento Go for Free | Log Out
Dashboard
Sales
Catalog
Mobile
Customers
Promotions
Newsletter
CMS
Reports
System
Get help for this page
Latest Message: Get More eCommerce Power with Magento Enterprise Read details
You have 1 critical, 4 major, 19 minor and 51 notice unread message(s). Go to messages inbox
Cache Storage Management
Flush Magento Cache
Flush Cache Storage
Select All | Unselect All | Select Visible | Unselect Visible | 8 items selected
Actions Refresh
Submit

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

After the system re-indexes data, a success message will be shown as follows:

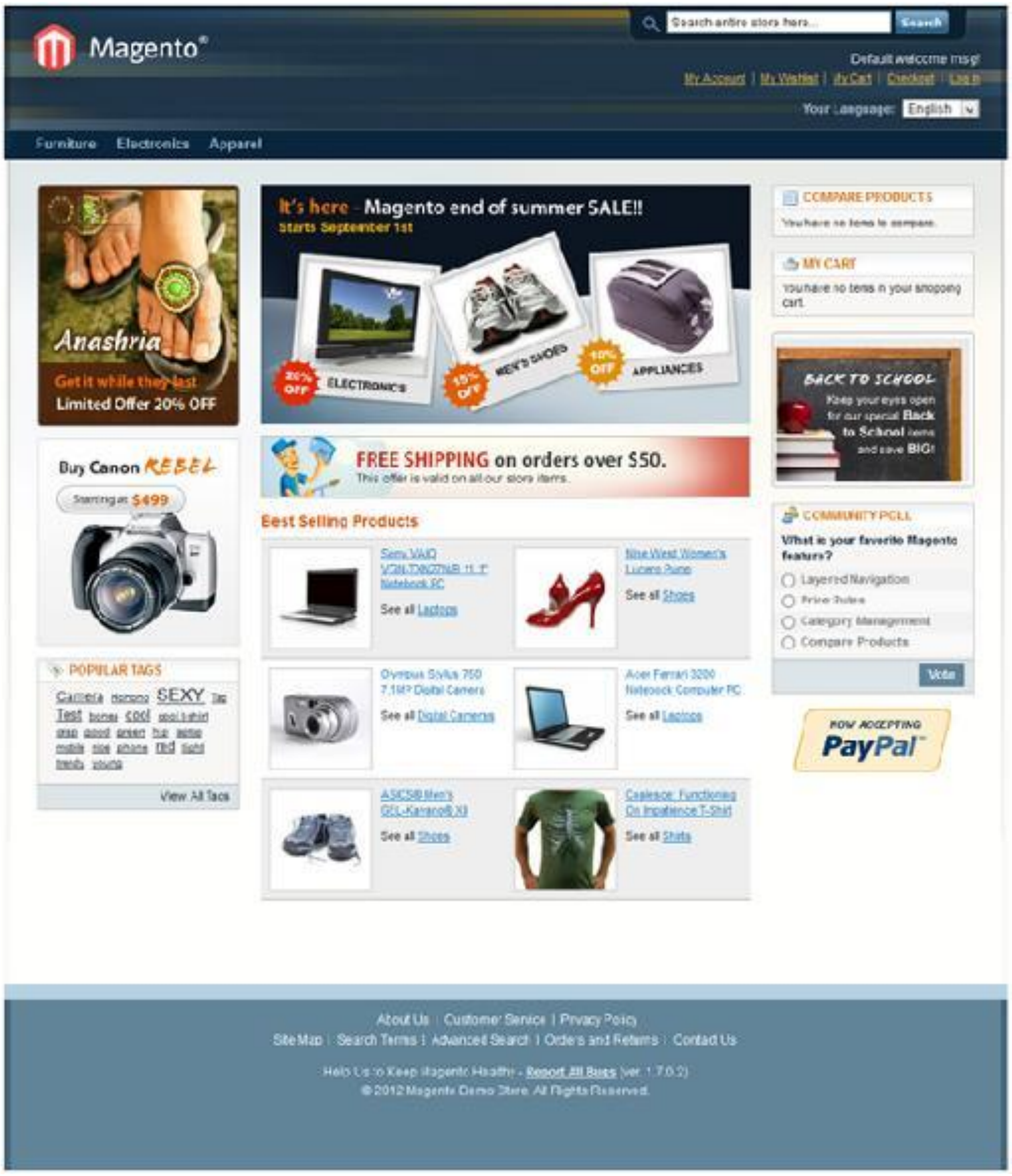

Magento Admin Panel
Global Record Search
Logged in as admin | Thursday, August 1, 2013 | Try Magento Go for Free | Log Out
Dashboard
Sales
Catalog
Mobile
Customers
Promotions
Newsletter
CMS
Reports
System
Get help for this page
Latest Message: Get More eCommerce Power with Magento Enterprise Read details
You have 1 critical, 4 major, 19 minor and 51 notice unread message(s). Go to messages inbox
Cache Storage Management
Flush Magento Cache
Flush Cache Storage
Select All | Unselect All | Select Visible | Unselect Visible | 0 items selected
Actions Refresh
Submit

8 cache type(s) refreshed.

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

2.6. Go to Frontend

Click on the “Go to Frontend” button in section 2.3 or type www.localhost.com/magento/ in your web browser. Now, you can see your Magento site:


Magento
Search entire store here...
Search
Default welcome msg!
My Account | My Wishlist | My Cart | Checkout | Log In
Your Language: English
Furniture
Electronics
Apparel
Anashria
Get it while they last
Limited Offer 20% OFF
It's here Magento end of summer SALE!!
starts September 1st
ELECTRONICS
MEN'S SHOES
APPLIANCES
FREE SHIPPING on orders over $50.
This offer is valid on all our store items.
Best Selling Products
Buy Canon REBEL
Starting at $499
POPULAR TAGS
View All Tags
COMPARE PRODUCTS
You have no items to compare.
MY CART
You have no items in your shopping cart.
BACK TO SCHOOL
COMMUNITY POLL
What is your favorite Magento feature?
Layered Navigation
Price Rules
Category Management
Compare Products
Vote
NOW ACCEPTING PayPal
About Us | Customer Service | Privacy Policy
Site Map | Search Terms | Advanced Search | Orders and Returns | Contact Us
© 2012 Magento Demo Store. All Rights Reserved.

III. SUMMARY

Above is the full guide to install webserver and set up a Magento website on your computer. This is the basic step for you to start with Magento development. Take some time try all the functions in backend and frontend before you turn to the next lesson.

Back to top

Lesson 02: Magento Architecture

I. OVERVIEW

Lessons 2, 3 and 4 will give you a general picture of Magento, so you are recommended to read them before moving to Lesson 5. From Lesson 5 on, you can use those lessons as references.

Magento has a wonderful architecture behind its system. The combination of Zend Framework and MVC (Model, View and Controller) Architecture makes Magento become a professional open-source e-commerce solution offering unprecedented flexibility and control.

1. Time: 3 hours

2. Content

- Magento Architecture
- Extension and module
- Block, Layout, Theme

II. CONTENT

1. Magento MVC Architecture

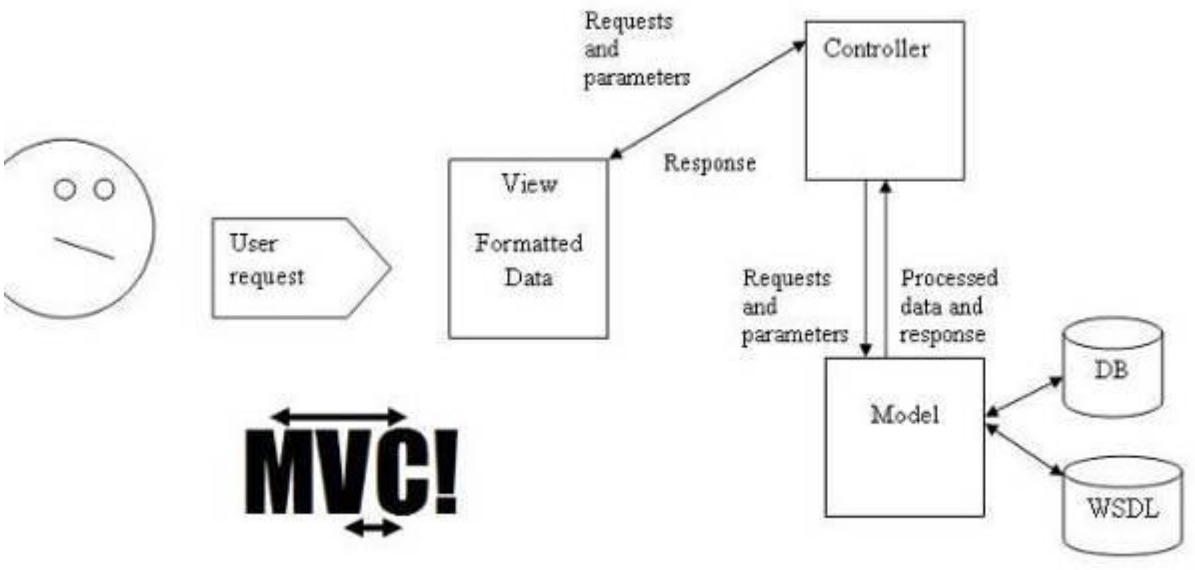
Magento is built on top of the Zend Framework, so the code base will be secure and scalable. There are many reasons for choosing the Zend Framework, but the main one is that Zend provides an object-oriented library of code with a committed company standing behind it.

Magento is built with three central tenets in mind:

- **Flexibility**: Each solution should be as unique as the business behind it. Magento code allows for seamless customizations.
- **Upgradeable**: By separating the core code from community and local customizations, Magento can be easily customized without losing the ability to upgrade.
- **Speed and Security**: The coding standards used by the developers follow best practices to maximize the efficiency of the software and provide a secure online storefront.

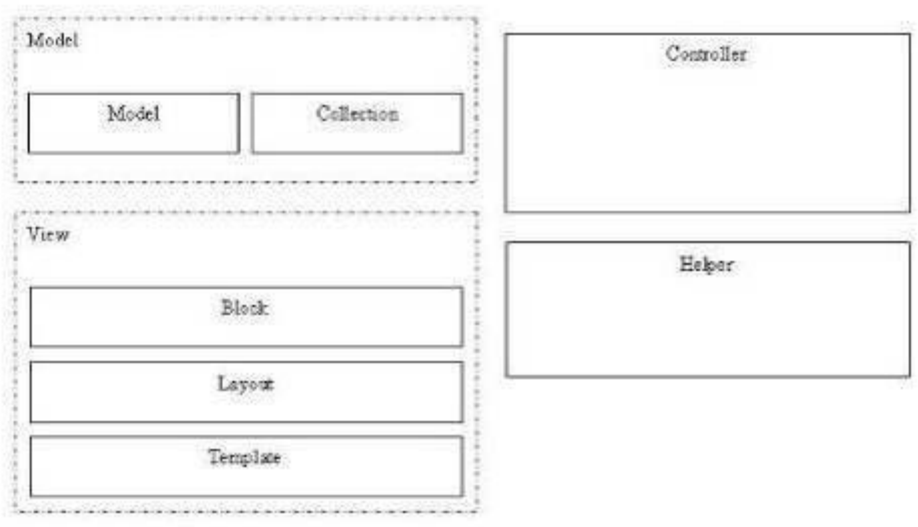
Magento architecture follows the well-known MVC architecture but it actually has some additions such as supporting large-scale web developments.

In MVC architecture, you have a set of modules that comes with Models, Views and Controllers to split up your codes and make code management easier and simpler. In the conventional MVC, you request the controller for a service, the controller uses models to get processed data and put forward the data to the view to give user the response and take another request which is carried out in the same fashion. This is an example of MVC:



Magento architecture has added a lot sub blocks to the above MVC architecture in order to handle bigger e-commerce system that can handle multiple sites/stores from the same back-end.

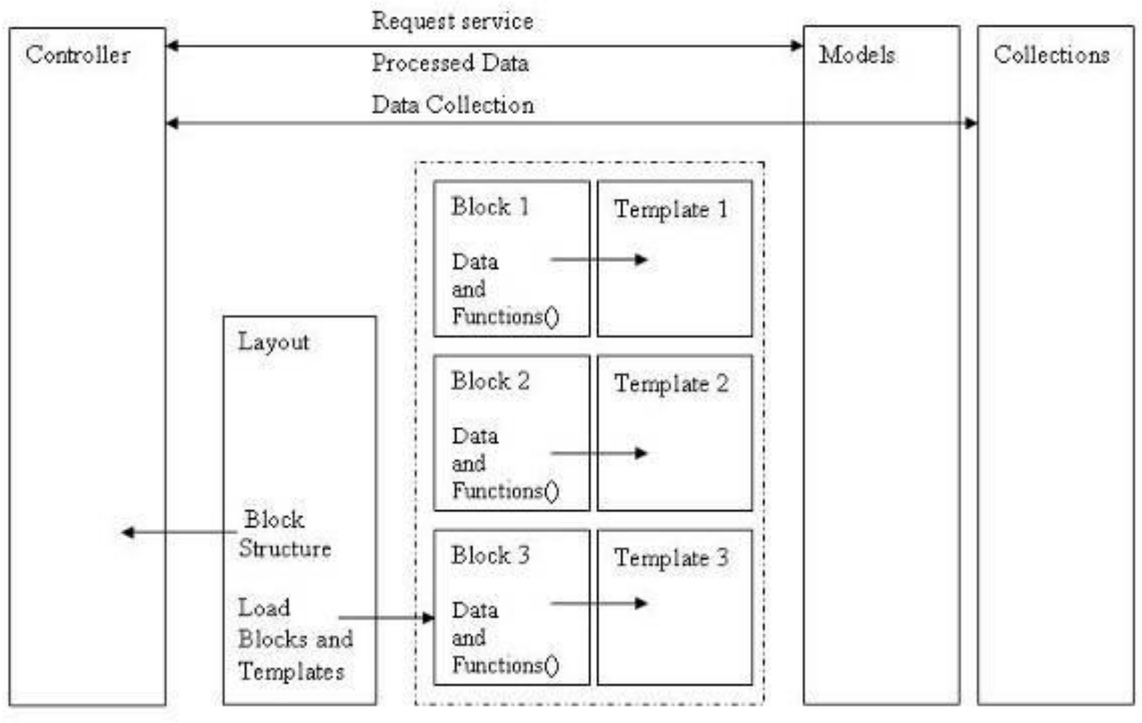
Magento Architecture View itself is broken into three parts, the model into two and you have controllers and helpers where helpers are module specific. If you think that why helpers are module specific, you should know that Magento helpers and in fact all the models and controllers extend Magento core controllers, models and helpers so the common features are there in the super class whereas you can also add module specific features which you need not share to make big helper classes. To start, see the image below:



You can see that Views has been divided into three parts. The templates are the plainly html codes usually saved as *phtmls* with php tags to prints data and do some basic loops and some javascript calls like our usual view would look like.

Next, Blocks, new concept to MVC, is simply used to lower the burden on central controller and make different views in a module more independent. This is important that websites now stand on a number of different blocks, some of which may be AJAX loaded and provide different services. Therefore, Views has its own controllers to ask or request processed data from Models and provide graphical ‘view’ through templates. Blocks holds all the data and functions that can be called from the view template and Block can have nested blocks. Therefore, our website will have a root block, a header block, a navigation block and contents and footer block.

Now, we might wonder what the role of the central controller is if every view comes with their own controller which can interact with models and helpers. That is where layout comes into play actually. In most of the definitions, they said that which blocks went with which template and defined which block was nested in which we needed the layout. Layouts can only be called by a central controller name; thus, the central controller has the layout that defines the sub controllers (blocks) and the templates it contains. The central controller and the helper provide service to the overall block set while the individual blocks provide service in email. Layout therefore simply provides a way to tell which the super block is and which are nested and how they are nested.



Finally, Models contains Models and collections. Models works as a service, it provides functionalities for various business calculations and data processing, whereas, the collection is used to provide functions to retrieve the Data.

2. Extension and module

2.1. Core, local and community

- **The core** – The core of Magento contains all the functionalities included in the downloaded version. The core code is a collection of modules developed or certified by the Magento core development team. Editing core files is not recommended and will disable the ability to upgrade Magento in the future.
- **The local** – Local extensions, which are customizations of Magento, reside only on a user’s local copy. These extensions will be placed in a local folder, so that they do not interfere with upgrades to the core code, and in order to differentiate them from community contributions. There are different types of extensions, but they will all reside in the same directory.

  Local extensions function just the same as core code does, only the directory is different.

- **The community** – When Community contributions are downloaded, they will reside in the community folder. Like local extensions, by keeping them separate from the core code, Magento storeowners are able to enjoy the additional functionality without compromising the ability to upgrade to future Magento versions.

2.2. Extension and module

*** Extension**

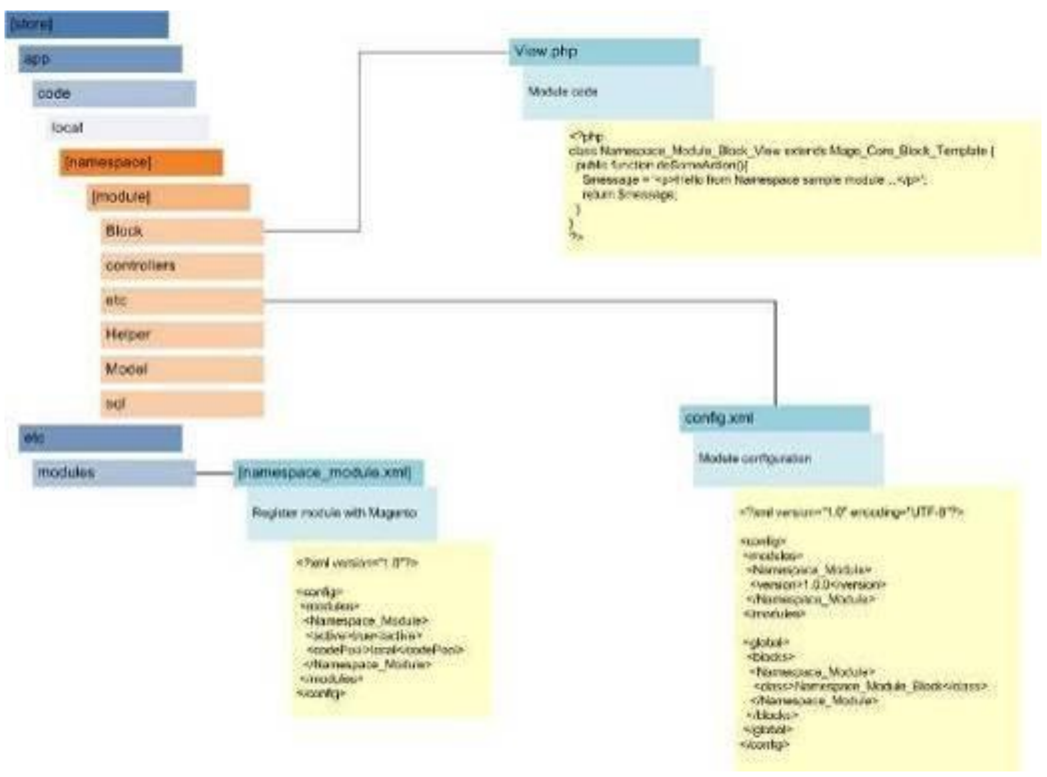
Extensions are exactly what they sound like. One or more files packaged together to extend the functionality of Magento. Strict terms and conditions prohibit extensions from modifying the core code, ensuring that any extended functionality does not prohibit you from upgrading when a new version of Magento is released.

Extensions can be either installed from the admin panel, or downloaded from Magento Connect. These processes will be covered later in the book, but look at the three types of extensions.

There are three types of extensions, and they will reside in one of the two locations described above.

*** Module**

A module is an extension that extends the features and functionality of Magento. You are probably familiar with the idea of modules from other software, but if not, some concrete examples of modules would be additional payment gateway integrations, or a featured items promotional tool. The image below show you the base structure of a module in Magento. You will learn about it closely in next lesson.



```php
<?php
class Namespace_Module_Block_View extends Mage_Core_Block_Template {
  public function doSomeAction(){
    $message = '<p>Hello from Namespace sample module ...</p>';
    return $message;
  }
}
?>
```

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

<config>
 <modules>
  <Namespace_Module>
   <active>true</active>
   <codePool>local</codePool>
  </Namespace_Module>
 </modules>
<config>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>

<config>
 <modules>
  <Namespace_Module>
   <version>1.0.0</version>
  </Namespace_Module>
 <modules>

 <global>
  <blocks>
   <Namespace_Module>
    <class>Namespace_Module_Block</class>
   </Namespace_Module>
  </blocks>
 <global>
</config>
```

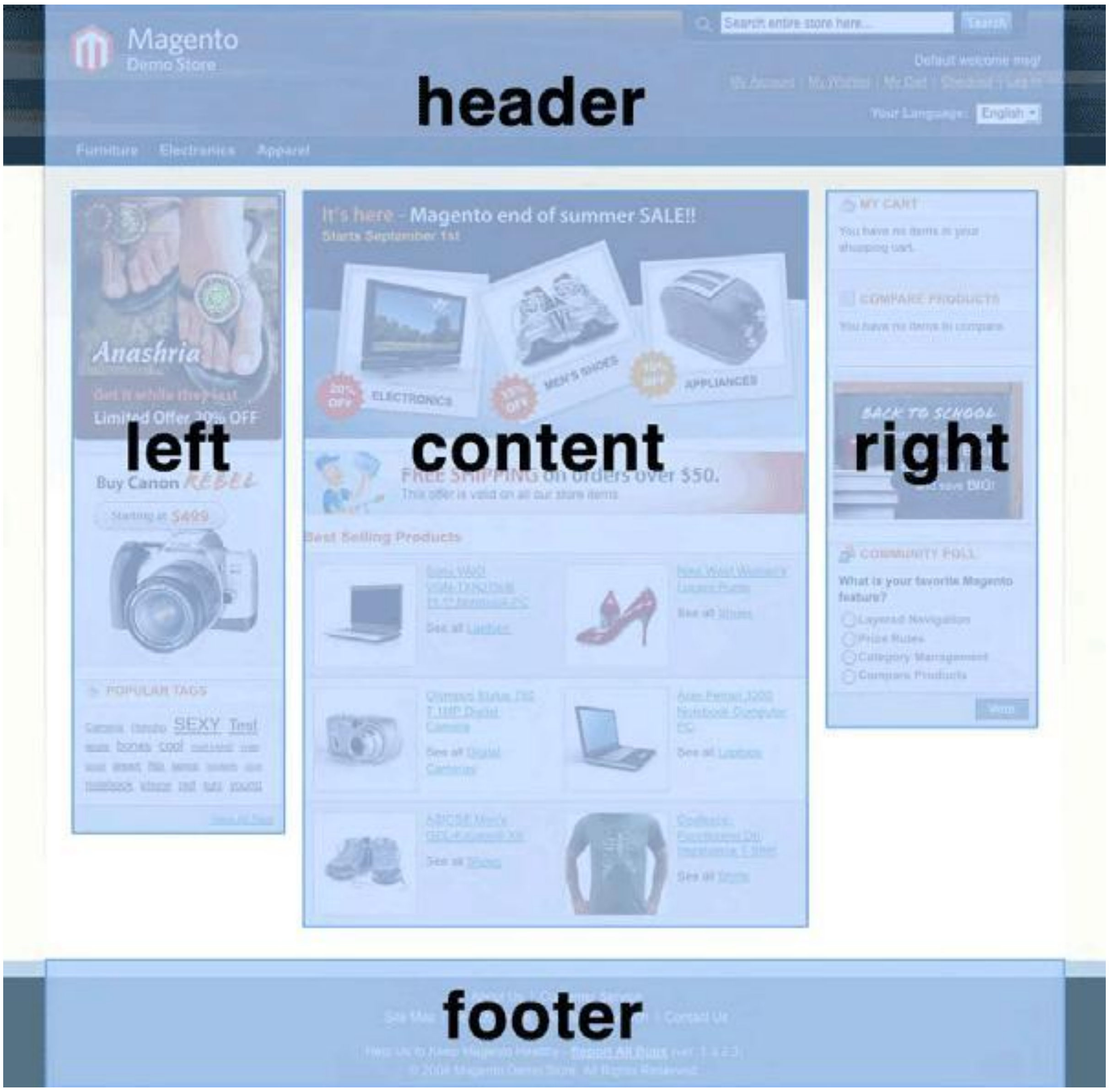
3. Block, Layout, Theme

3.1. Block

In Magento, there are two kinds of block: Structure block and Content block

Structural Block: These are blocks created for the sole purpose of assigning visual structure to a store page. Structural block is used to separate theme into sections. Look at the three-column layout. The structural blocks are

- Header
- Left
- Content
- Right
- Footer



Structural blocks of layout three-column

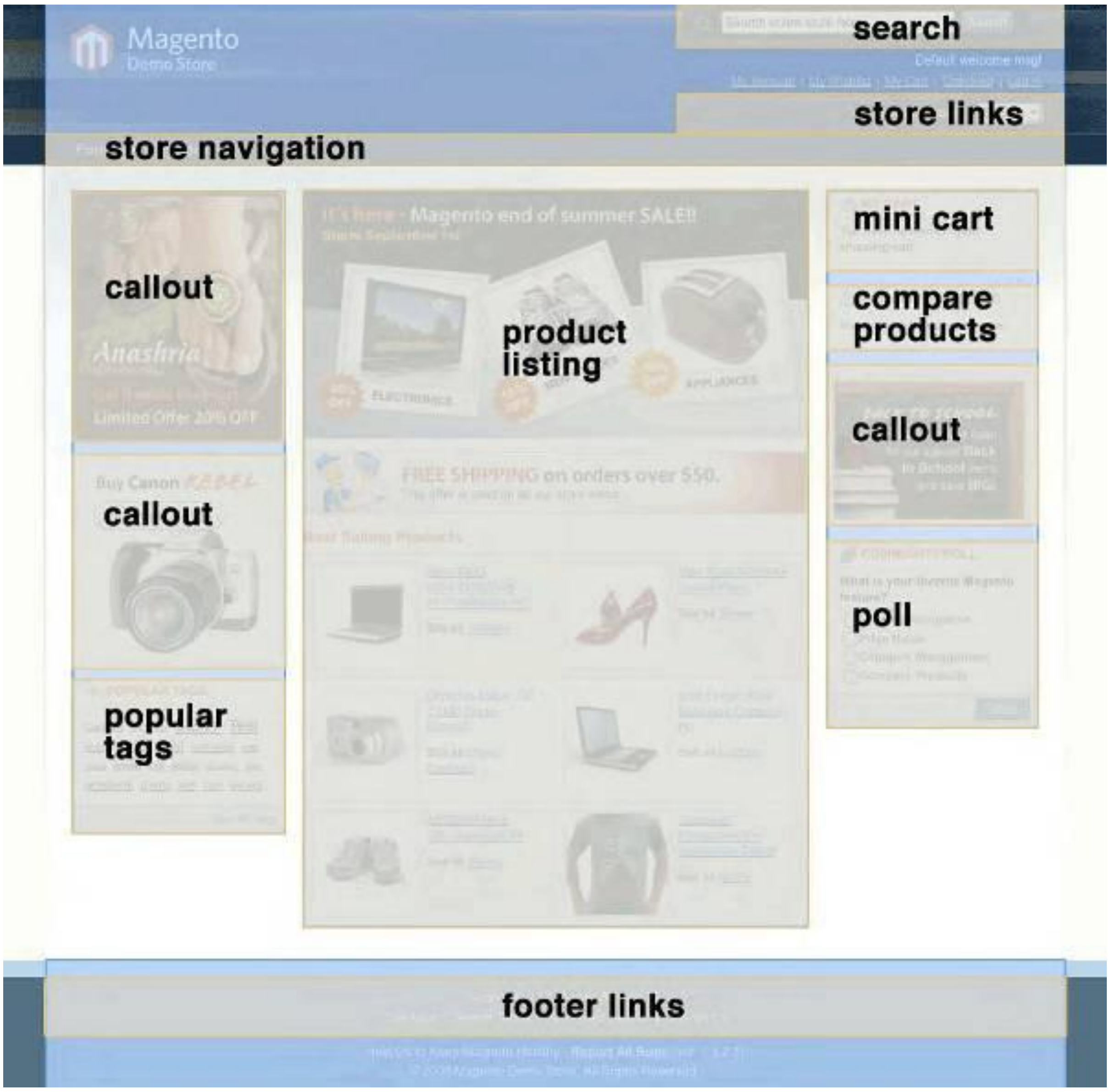
Content Block: These blocks produce the actual content inside each structural block. They are representations of each feature functionality in a page and employ template files to generate the (X)HTML to be inserted into its parent structural block.Take the Right column. The content blocks should be:

- Mini cart

- Recently viewed product

- Newsletter subscription block

- Poll



Content blocks in each of structural blocks

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

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

3.2. Layout

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

A layout XML file is structured as follows:

```
<layout version="0.1.0">
       <default>
              <reference name="header">
            <block type="page/html_header"  name="header"
as="header">
             <block type="page/template_links" name="top.links"
                          as="topLinks"/>
             <block type="page/switch" name="store_language"
                    as="store_language"

  template="page/switch/languages.phtml"/>
             <block type="core/text_list" name="top.menu"
as="topMenu"                     translate="label">
             <label>Navigation Bar</label>
             </block>
            </block>
           </reference>
          </default>
        </layout>
```

In the code above we have:

- **<default>:** The handler for the URL, in this case default will load no matter what other handler is being initialized. If you replace the <default> tag by <module_controller_action> tag, the blocks put in will load when the action is called.
- **<reference>**: The reference structure which calls the blocks in our theme.
- **<block>**: A content block which defines the type of block and the template will process the block’s outgoing data in the system. Type parameter defines the module class which contains the function of the block. There are two ways to put a content block into a structural block. Before = “-” and after = “-” are the parameters that help you put the content block into position you want.

In addition, Magento uses actions within blocks for functions which need to process data such as adding CSS stylesheeds:

```
<action method="addCss">
        <stylesheet>css/styles.css</stylesheet>
   </action>

   <action method="addJs">
              <script>varien/js.js</script>
   </action>

   <action method="addItem">
              <type>skin_css</type>
              <name>css/styles-ie.css</name>
              <params/>
              <if>lt IE 8</if>
   </action>
```

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

3.3. Theme

A theme is a combination of layout, template and/or skin file(s) that create the visual experience of your store. Magento is built with the capacity to load multiple themes at once, so it distinguishes themes into two types:

- **Default Themes**: Every interface comes with a theme called 'default' which is the main theme of an interface. When you assign an interface to your store, the application automatically looks for this theme 'default' and loads it to the front-end. In order to customize your store design, you can either modify this theme, or create a non-default theme in addition and load it alongside the default. The default theme must contain all the required layouts, templates and skins to run a store error-free and hence is the lowest theme in the theme hierarchy.
- **Non-Default Themes**: A non-default theme can contain as many theme files as you want. This type of theme is intended for creating temporary seasonal design changes to a store without having to create a new default theme. By creating a few images and

updating some CSS, you can easily turn your store from a real bore to a stand-out seasonal Christmas store.

Components of a theme:

- **Layouts**: Layouts are basic XML files that define the block structure for different pages, as well as controlling the META information and page encoding. Layout files are separated on a per-module basis with every module bringing with it its own layout file.
- **Templates**: Templates are PHTML files that contain (X)HTML markups and any necessary PHP tags to create the logic for the visual presentation of information and features.
- **Skins**: Skins are block-specific JavaScript, CSS and image files that compliment your (X)HTML.

III. SUMMARY AND REFERENCE

That is the end of Lesson 2. Now, it is sure that you can figure out:

- How Model, View and Controller can work together in a MVC Architecture;
- How the core, the local and the community are different;
- The functions of Extension and Module;
- Where Block, Layout and Theme lie in a Magento website.

If you want to dig deeper into this topic, you can read the following materials:

- Jamie Huskisson, 2000. *Magento 1.3: PHP Developer's Guide*. United Kingdom. Packt Publishing Ltd.

- Magento PHP Blog, 2010. *Magento Architecture*. [Online] Available at http://magentophp.blogspot.com/2010/08/magento-architecture.html

- Magentocommerce, n.d.. *Chapter 1: What is Magento?* [Online] Available at http://www.magentocommerce.com/wiki/welcome_to_the_magento_user_s_guide/chapter_1

Back to top

Lesson 03: Magento Folder and Data Structure

I. INTRODUCTION

1. Time: 2 hours

2. Content:

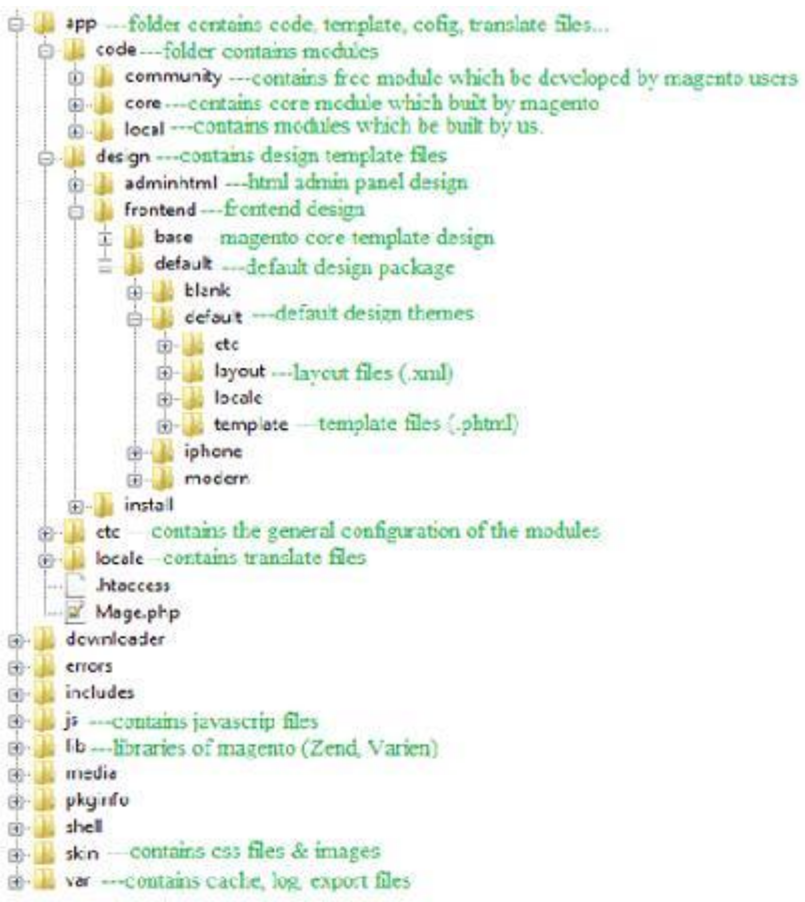
- Magento Folder structure

- Database of some main Magento modules (customer, product, sales)

II. CONTENT

1. Magento Folder Structure

Magento is constructed based on the principle of object-oriented programming under MVC architecture. The Magento code system is stored in the form of dispersions in order to increase extension ability for the system. The folder system is stored as follows:

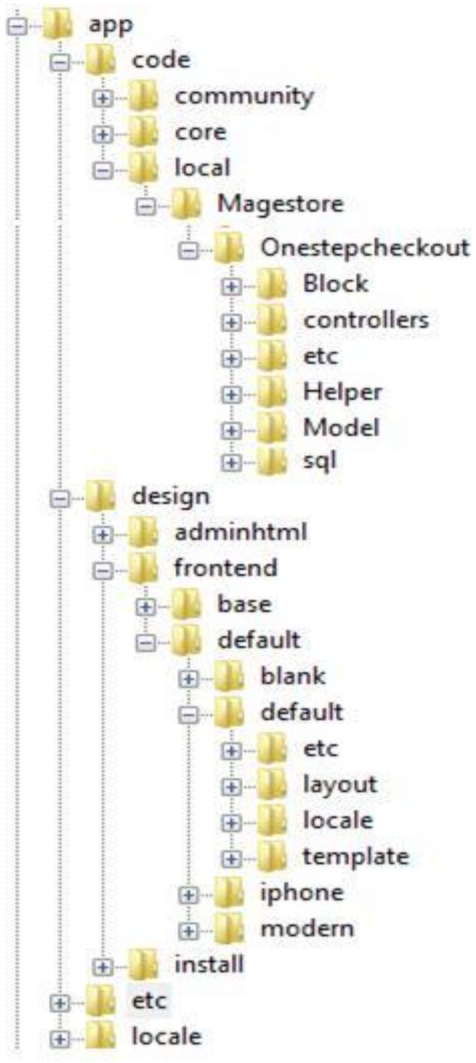


```
app ---folder contains code, template, cofig, translate files...
  code ---folder contains modules
    community ---contains free module which be developed by magento users
    core ---contains core module which built by magento
    local ---contains modules which be built by us.
  design ---contains design template files
    adminhtml ---html admin panel design
    frontend ---frontend design
      base    magento core template design
      default ---default design package
        blank
        default ---default design themes
          etc
          layout ---layout files (.xml)
          locale
          template ---template files (.phtml)
        iphone
        modern
    install
  etc --- contains the general configuration of the modules
  locale ---contains translate files
  .htaccess
  Mage.php
downloader
errors
includes
js ---contains javascrip files
lib ---libraries of magento (Zend, Varien)
media
pkginfo
shell
skin ---contains css files & images
var ---contains cache, log, export files
```

2. A Module Structure in Magento

A module is an extension which extends the Magento's features and functionalities. You are probably familiar with the idea of modules from other software, but if not, some concrete examples of modules would be additional payment gateway integrations, or a featured items promotional tool.

Main folders in Magento module are:



Code folders of a module are usually written in the folder app/code/local and include:

- **<Namespace>**

  <Modulename>

    - Block: the folder containing blocks
    - Helper: the folder containing helper files

  - controllers: the folder containing controllers of the module
  - etc: the folder containing configuration files for the module
  - Model: the folder containing models of the module
    sql: containing data files for the module

- **Namespace**: A unique name identifies your company or organization. The intent is that each member of the worldwide Magento community will use their own Package names when creating modules in order to avoid colliding with another user's code.
- **Block**: The drivers behind the arrangement of Magento templates.
- **Controllers**: They contain controlling layers of application flow. They receive users' requests via HTTP header as input and then forward those requests to the layers directly in charge of request processing.
- **etc**: It contains configuration files of a module (XML file). Based on the configuration files, it can set up or overwrite the old settings just by placing the correct XML tags.
- **Helper**: It contains utility methods that allow you to carry out common tasks on objects and variables.
- **Model**: It contains layers that provide data and services related to data and business logic. These classes work directly with the database and provide data for other components.
- **sql**: It contains files of installation and database update of the module database.

In addition to code folders, there are folders containing files of interface, configuration, language, etc.

- **Layout** (*/app/design/frontend/packagename/themesname/layout)*: Layouts are basic XML files that define the block structure for different pages, as well as controlling the META information and page encoding. Layout files are separated on a per-module basis, with every module bringing with it its own layout file.
- **Template** (*/app/design/frontend/packagename/themesname/template)*: Templates are PHTML files that contain (X)HTML markups and any necessary PHP tags to create the logic for the visual presentation of information and features.

- **Skins** (*/skin/frontend/packagename/themesname)*: Default Skins are block-specific JavaScript and CSS and image files that compliment your (X)HTML.

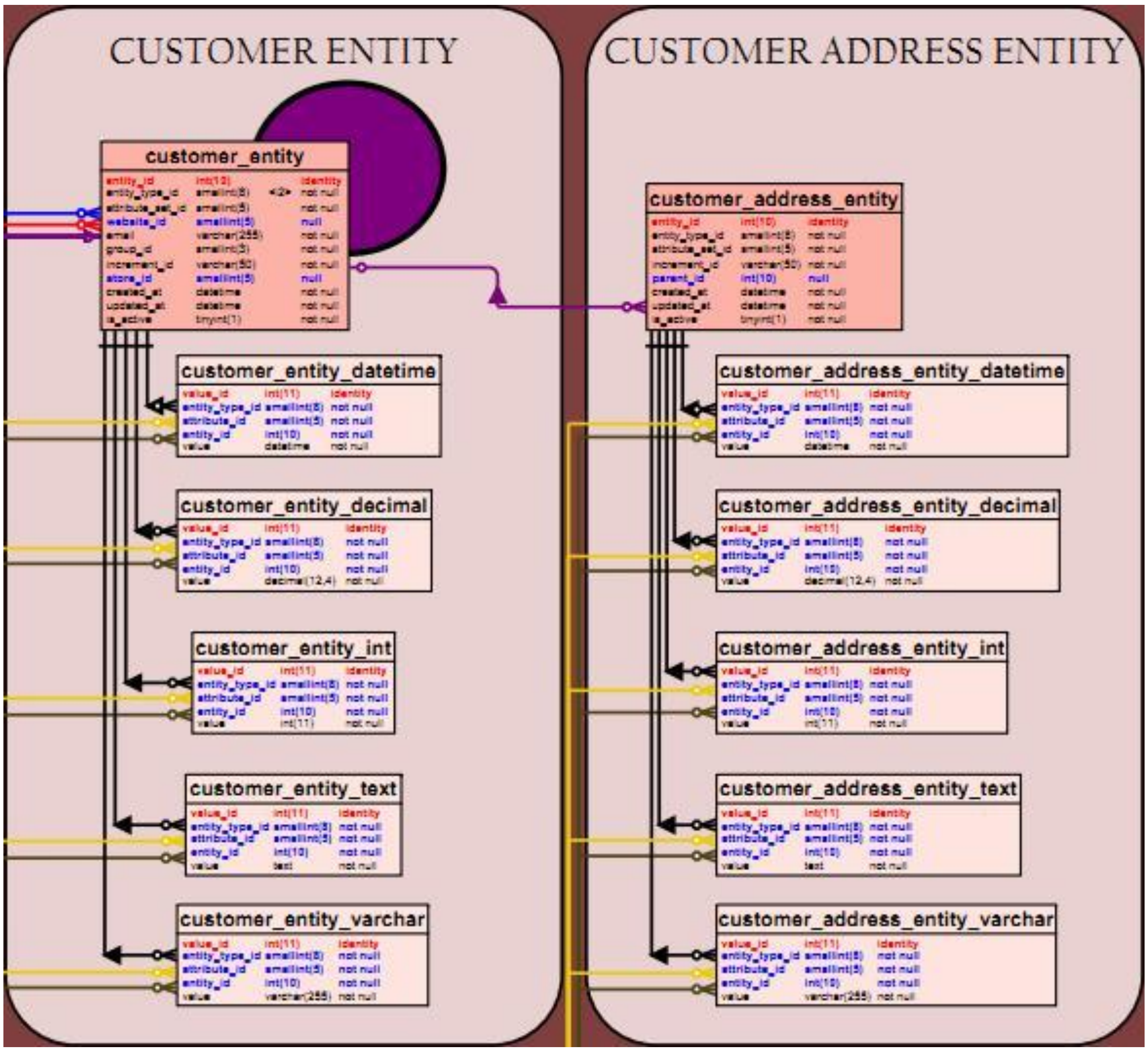
2. Data System in Magento

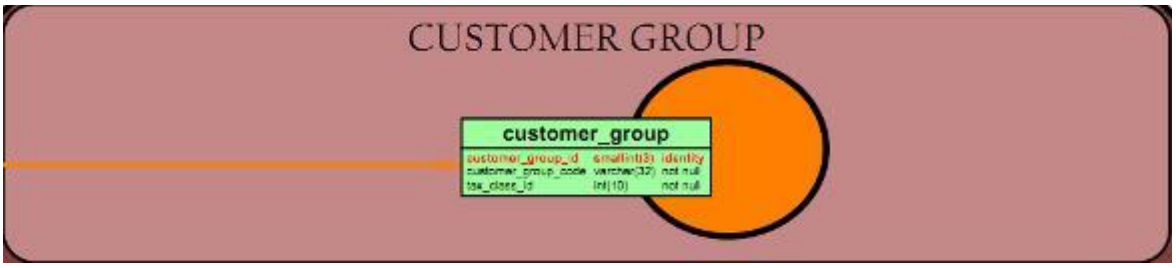
Unlike other platforms, Magento uses EAV database model. The advantage of this model is its flexibility to use properties, which is very important to an e-commerce website. This model will be introduced in detail in Lesson 19.

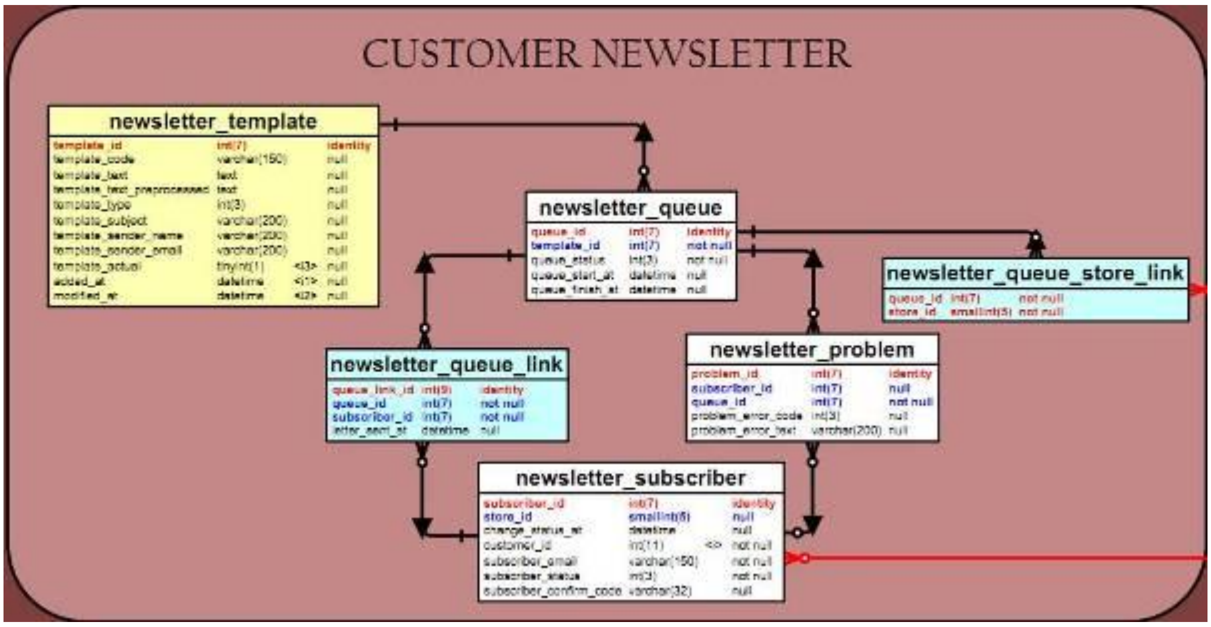
In this section, we will learn about the database of some main modules of Magento including *Customer, Catalog and Sales*.

2.1. Customer

Customer database in Magento is stored under EAV model and includes some following tables:


CUSTOMER ENTITY
CUSTOMER ADDRESS ENTITY
customer_entity
entity_id int(10) identity
entity_type_id smallint(8) <2> not null
attribute_set_id smallint(5) not null
website_id smallint(5) null
email varchar(255) not null
group_id smallint(3) not null
increment_id varchar(50) not null
store_id smallint(5) null
created_at datetime not null
updated_at datetime not null
is_active tinyint(1) not null
customer_entity_datetime
customer_entity_decimal
customer_entity_int
customer_entity_text
customer_entity_varchar
customer_address_entity
entity_id int(10) identity
entity_type_id smallint(8) not null
attribute_set_id smallint(5) not null
increment_id varchar(50) not null
parent_id int(10) null
created_at datetime not null
updated_at datetime not null
is_active tinyint(1) not null
customer_address_entity_datetime
customer_address_entity_decimal
customer_address_entity_int
customer_address_entity_text
customer_address_entity_varchar
value_id int(11) identity
entity_type_id smallint(8) not null
attribute_id smallint(5) not null
entity_id int(10) not null
value datetime not null
value decimal(12,4) not null
value int(11) not null
value text not null
value varchar(255) not null


CUSTOMER GROUP
customer_group
customer_group_id smallint(3) identity
customer_group_code varchar(32) not null
tax_class_id int(10) not null



The Customer_entity table stores main information about customers, which includes:

- entity_id: primary key
- email: customer’s email
- create_at, update_at: time created, time last updated for customer
- is_active: active status
- website_id, store_id, group_id: foreign keys linking to corresponding tables: core_website, core_store_id, customer_group
- entity_type_id: primary key of table eav_entity_type
- attribute_set_id: primary key of table eav_attribute_set

The tables customer_entity_datetime, customer_entity_decimal, customer_entity_int, customer_entity_text, customer_entity_varchar contain value of the attributes categorized by data type. In these tables, these are some noteworthy fields:

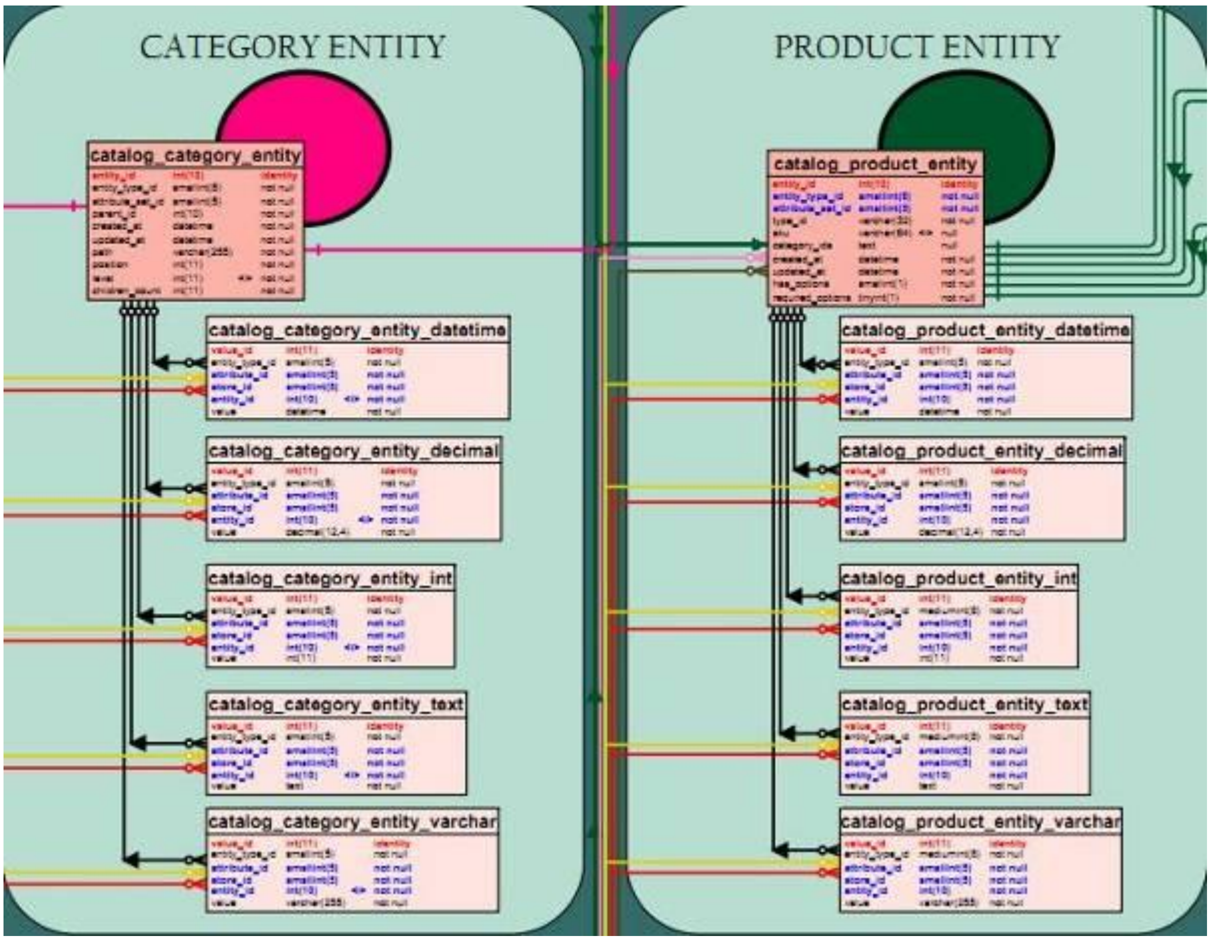
- entity_type_id: primary key of table eav_entity_type

- attribute_id: primary key of table eav_attribute.
- entity_id: primary key of table customer_entity

The tables for customer address have a similar storage method with customer_address_entity as the primary tables; moreover, there is the field parent_id to make a link to table customer_entity.

2.2. Catalog

In Magento, the database of module catalog can best reflect the way to use EAV model. The data of this module is divided into two sections: category and product.


CATEGORY ENTITY
PRODUCT ENTITY
catalog_category_entity
catalog_category_entity_datetime
catalog_category_entity_decimal
catalog_category_entity_int
catalog_category_entity_text
catalog_category_entity_varchar
catalog_product_entity
catalog_product_entity_datetime
catalog_product_entity_decimal
catalog_product_entity_int
catalog_product_entity_text
catalog_product_entity_varchar

The way to store the category table is similar to that of customer table, the primary table is catalog_category_entity and the secondary tables are ones with prefix such as catalog_category. It is noted that this table is the field parent_id to store information about the relationship between categories (parent category).

The table catalog_category_product shows the relationship between product and category, i.e. which product contained in which category.

Database for product is quite big, which is necessary for an e-commerce platform such as Magento. Like customer, the primary storing table - catalog_product_entity - includes the following tables:

- entity_id (primary key)
- create_at, update_at: time created, time last updated for customer
- entity_type_id: primary key of table eav_entity_type
- attribute_set_id: primary key of table eav_attribute_set
- Type_id: product type
- sku: product code

Tables with prefix catalog_product_entity store values of product attributes.

The table catalog_product_link expresses the relationships between products, which relationship types are stored in catalog_product_link_type, the values of relationship attributes are stored in the other table with corresponding prefix.

The table catalog_product_option contains information of options that product may have. This table is linked to catalog_product_bundle_option.

The table catalog_product_website show the relationship between product and website, product in Magento is created by separate websites.

Additionally, tables catalog_category_flat_store_x and catalog_product_flat_1 (x=1,2,3….) show the relationship between category, product and store. However, this is a disadvantage of Magento when using EAV model. Whenever a new store is created, other two similar tables are auto-generated, which is bad for websites with large number of stores and server which cannot serve memory requirements.

2.3. Sales

The database for module Sales stores necessary information for purchasing including: Quote, Order, Invoice, Shipment, Creditmemo. Unlike Customer and Catalog, tables in module Sales are stored in relationship database model.

Quote: Contains order information when a customer or admin creates a shopping cart. Table sales_flat_quote is a primary table. Tables sales_flat_quote_address, sales_flat_quote_item and sales_flat_quote_payment store detailed information about order address, items in the order, payment methods.

- **Order**: Contains order information after the customer confirms the order. Like Quote, primary table is sales_flat_order, secondary tables store other necessary information.
- **Invoice**: Invoices information after the order is processed. Primary tables sales_flat_invoice and table sales_flat_invoice_item store detailed information of the items in the order and table sales_flat_invoice_grid stores necessary information for report and analysis.
- **Shipment & Creditmemo**: These tables are designed like invoice. They contain order information after admin processes (shipment or cancellation) primary tables sales_flat_shipment and sales_flat_creditmemo. The system details are like that of Invoice.

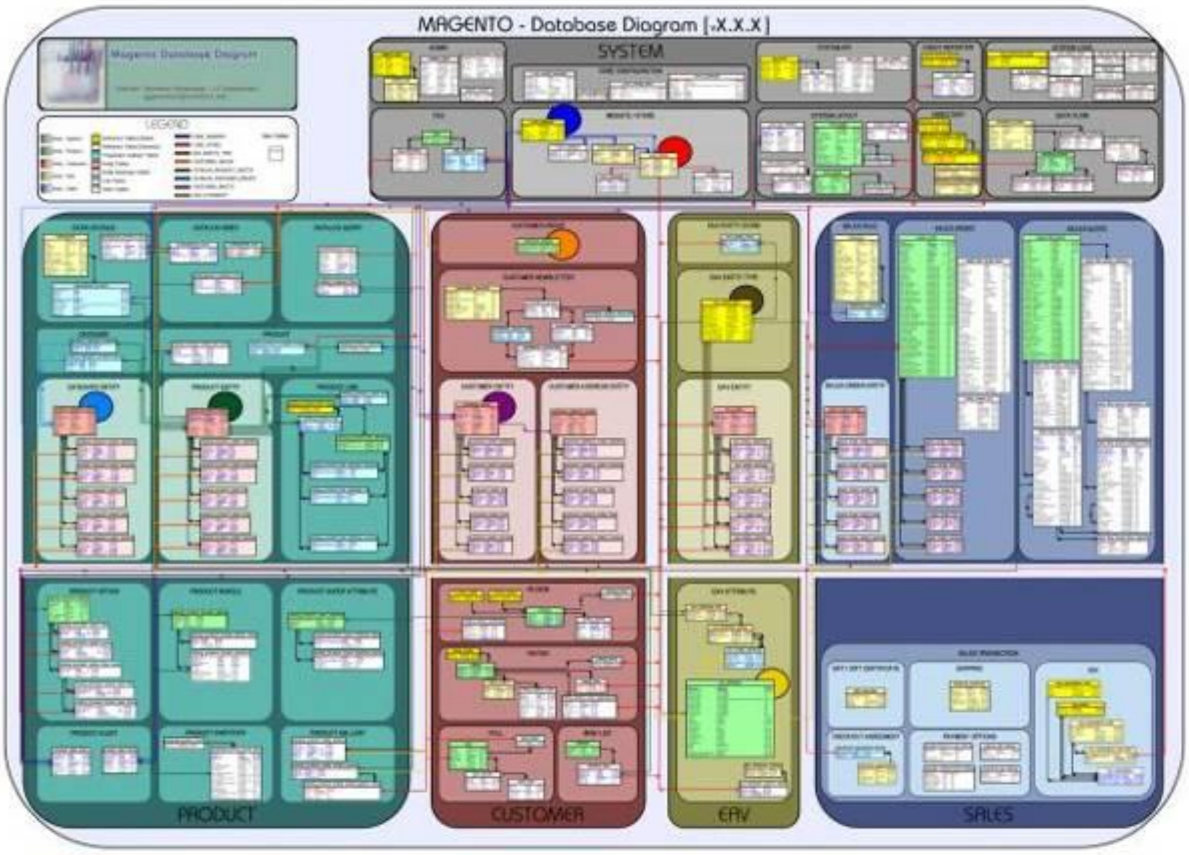
There are also other tables:

- sales_order_status: Information of order status list
- sales_order_tax: Information of cases that tax on the order
- salesrule: Information of conditions, promotion campaigns for each order
- sales_recurring_profile: …

III. SUMMARY AND REFERENCES

We have learnt the overall picture of Magento folders and data system, their positions and functions.

This picture is a visual illustration of the data system in Magento:


MAGENTO - Database Diagram [.X.X.X]
Magento Database Diagram
LEGEND
SYSTEM
PRODUCT
CUSTOMER
EAV
SALES

[Back to top]

Lesson 04: Configuration and Naming in Magento

I. OVERVIEW

1. Time: 3 hours

2. Content:

- Module configuration in config.xml
- Naming classes and files in Magento
- Calling methods to create class objects by config and through Mage class

II. CONTENT

1. Module configuration in config.xml

After the module list is loaded, the system will search for file config.xml in the module directory *(app/code/[codepool]/[NameSpace]/[ModuleName]/etc/config.xml)* to read the configuration of that module. The configurations are placed in tags <config></config>. In <config> tags, there are the following tags:

- **modules**: <modules></modules> tags, including configurations for module such as active, depends and version.

```
<modules>
    <[NameSpace_ModuleName]>
            <active>[true|false]</active>
        <codePool>[core|community|local]</codePool>
        <depends>
                    <[NameSpace_ModuleName] />
        </depends>
        <version>(version_number)</version>
    </[NameSpace_ModuleName]>
</modules>
```

- **global**: <global></global> tags, including main configuration for models, resources, blocks, helpers, fieldsets, template, events, eav_attributes, custom_variables. The configurations in this section include basic configuration for module, configuration about folder containing class files for model, block, helper and events that are called (both frontend and admin), configurations for email template and so on.

```
<global>
<models>
        <[ModuleName]>
            <class>[ClassName_Prefix]</class>
<resourceModel>[ModuleName]_[resource_model_type]</resourceModel>
            <[ModuleName]_[resource_model_type]>
                <!-- definition -->
            </[ModuleName]_[resource_model_type]>
            <rewrite><!-- definition --></rewrite>
        </[ModuleName]>
    </models>
    <resources>
        <[ModuleName]_setup><!-- definition --></[ModuleName]_setup>
        <[ModuleName]_read><!-- definition --></[ModuleName]_read>
        <[ModuleName]_write><!-- definition --></[ModuleName]_write>
    </resources>
    <blocks>
        <[ModuleName]>
            <class>[ClassName_Prefix]</class>
    </[ModuleName]>
</blocks>
<helpers>
        <[ModuleName]>
            <class>[ClassName_Prefix]</class>
        </[ModuleName]>
    </helpers>
    <fieldsets>
        <[page_handle]>
            <[field_name]> ><!-- definition --></[field_name]>
        </[page_handle]>
    </fieldsets>
    <template>
        <email>
            <[email_template_name] module="[ModuleName]"
                    translate="[label][description]">
                <!-- definition -->
            <[/email_template_name]>
        </email>
    </template>
<events>
        <[event_name]>
```

```
            <observers><!-- observers --></observers>
        </[event_name]>
    </events>
    <eav_attributes><!-- definition --></eav_attributes>
    <[ModuleName]><!-- custom config variables --></[ModuleName]>
</global>
```

- **admin**: <admin></admin> tags including attributes, field sets, routers configuration. In this config tag, we usually note about configuration for router of admin.

```
<admin>
<attributes>
    <[attribute_name] />
    <attributes>
    <fieldsets><!-- definition --></fieldsets>
    <routers>
        <[ModuleName]>
            <use>[standard|admin|default]</use>
            <args>
                <module>[NameSpace_ModuleName]</module>
                <frontName>[frontname]</frontName>
            </args>
        </[ModuleName]>
<!-- or --><[ModuleName]>
            <args>
                <modules>
                    <[NameSpace_ModuleName]
            before="[AnotherNameSpace_ModuleName]">
                        [New_ClassName]
                    </[NameSpace_ModuleName]
                </modules>
            </args>
        </[ModuleName]>
    </routers>
</admin>
```

- **adminhtml**: <adminhtml></adminhtml> tags, including configurations of modules in the admin area. The tags are usually layout, translate and events. These configurations only take effect in backend in Magento.

```
<adminhtml>
    <events>
```

```
        <[event_name]>
            <observers><!-- observers --></observers>
        </[event_name]>
    </events>
    <global_search>
        <products>
            <class>[ModuleName]/search_catalog</class>
            <acl>catalog</acl>
        </products>
        <customers>
            <class>adminhtml/search_customer</class>
            <acl>customer</acl>
        </customers>
        <sales>
            <class>adminhtml/search_order</class>
            <acl>sales</acl>
        </sales>
    </global_search>
    <translate>
        <modules>
            <[NameSpace_ModuleName]>
                <files>
                <default>(name_of_translation_file.csv)
                </default>
                </files>
            </[NameSpace_ModuleName]>
        </modules>
    </translate>
    <layout>
        <updates>
            <[ModuleName]>
                <file>[name_of_layout_file.xml]</file>
            </[ModuleName]>
        </updates>
    </layout>
    <[ModuleName]><!-- custom config variables --></[ModuleName]>
</adminhtml>
```

- **frontend**: <frontend></frontend> tags containing active configuration of module on frontend. It includes secure_url, events, routers, translate and layout tags. These configurations only take effect on frontend area.

```
<frontend>
    <secure_url>
        <[page_handle]>/relative/url</page_handle>
    </secure_url>
    <events>
        <[event_name]>
            <observers><!-- observers --></observers>
        </[event_name]>
    </events>
    <routers>
        <[ModuleName]>
            <use>[standard|admin|default]</use>
            <args>
                <module>[NameSpace_ModuleName]</module>
                <frontName>[frontname]</frontName>
            </args>
        </[ModuleName]>
    </routers>
    <translate>
        <modules>
            <[NameSpace_ModuleName]>
                <files>
                    <default>
                        (name_of_translation_file.csv)
                    </default>
                </files>
            </[NameSpace_ModuleName]>
        </modules>
    </translate>
    <layout>
        <updates>
            <[ModuleName]>
                <file>(name_of_layout_file.xml)</file>
            </[ModuleName]>
        </updates>
    </layout>
</frontend>
```

- **default**: <default></default> tags, containing default configurations of module in the whole stores/websites.

- **stores**: <stores></stores> tags, containing configurations for individual stores. They include *<(store_code)>* config tag for each store.
- **websites**: <websites></websites> tags, containing configurations for individual websites. They include <(website_code)> config tag for each website.

2. Naming classes and files in Magento

2.1. Controller

- File name: [Namecontroller]Controller.php in directories
  - *app/code/[Codepool/[Namespace]/[ModuleName]/controllers/*
  - *app/code/[Codepool]/[Namespace]/[ModuleName]/controllers/Adminhtml/*

For example:

app/code/local/Magestore/Lesson04/controllers/Lesson04Controller.php
or

app/code/local/Magestore/Bigdeal/controllers/BigdealController.php
or *app/code/local/Magestore/Bigdeal/controllers/Adminhtml/BigdealController.php*

- Class name: [NameSpace]_[Module]_[Namecontroller]Controller and

  [NameSpace]_[Module]_Adminhtml_ [Namecontroller]Controller.

For example:

class Magestore_Lesson04_IndexController extends Mage_Core_Controller_Front_Action{}
or

class Magestore_Lesson04_ Adminhtml_ReportController extends Mage_Adminhtml_Controller_Action{}

2.2. Helper

- File name: [Namehelper].php in directory

  app/code/[Codepool]/[Namespace]/[Module]/helper/

For example:

app/code/local/Magestore/Lesson04/Helper/Data.php
app/code/local/Magestore/Lesson04/Helper/Lesson04.php

- Class name: [NameSpace]_[Module]_Helper_[Filehelpername]

For example:

Class Magestore_Lesson04_Helper_Lesson04 extends Mage_Core_Helper_Abstract{}

2.3. Model:

- File name:
    - Model: [Namemodel].php in directory app/code/[Codepool]/[Namespace]/[Module]/Model/
    - Mysql4: [Namemodel].php in directory app/code/[Codepool]/[Namespace]/[Module]/Model/Mysql4
    - Collection: Collection.php in directory app/code/[Codepool]/[Namespace]/[Module]/Model/Mysql4/([Modelname])

- Class name:
    - Model: [Namespace_Module]_Model_[Modelname]

      For example:

      class Magestore_Lesson04_Model_Lesson04 extends

      Mage_Core_Model_Abstract{}
    - Mysql4: [Namespace_Module]_Model_Mysql4_[Modelname]

      For example:

      class Magestore_Lesson04_Model_Mysql4_Lesson04 extends

      Mage_Core_Model_Mysql4_Abstract{}
    - Collection: [Namespace_Module]_Model_Mysql4__[Modelname]_Collection

      For example:

      class Magestore_Lesson04_Model_Mysql4_Lesson04_Collection extends

      Mage_Core_Model_Mysql4_Collection_Abstract{}

2.4. SQL

- File name:

- Installation file name: mysql4-install-[start-version].php in directory app/code/local/[NameSpace]/[Module]/sql/

  For example:

  app/code/local/Magestore/Lesson04/sql/mysql4-install-0.1.0.php

- Upgrade file name: mysql4-upgrade-[old-version]-[new-version].php

  For example:

  app/code/local/Magestore/Lesson04/sql/mysql4-upgrade-0.1.0-0.1.1.php

2.5. Block

- File name: [blockname].php in directories

  app/code/local/[NameSpace]/[Module]/Block/

  and

  app/code/local/[NameSpace]/[Module]/Block/Adminhtml/

For example:

App/code/local/Magestore/Lesson04/Block/Lesson04.php
App/code/local/Magestore/Lesson04/Block/ Adminhtml/Lesson04.php

3. Calling methods to create class objects by config and through class Mage

3.1. Popular call functions in class Mage

```
-   Mage::getModel ($modelClass = '',$arguments = array())
Retrieve model object.

-   Mage:: getModuleDir($type,$moduleName)
Retrieve application module absolute path.

-   Mage:: getResourceModel($modelClass,$ arguments = array())
Retrieve object of resource model.

-   Mage:: getSingleton($modelClass = '',array  $arguments =
array())
Retrieve model object singleton.
```

```
-    Mage::getStoreConfig ($path,$id = null  )

-    Mage:: getUrl ($route = '', $params = array())
Generate url by route and parameters.

-    Mage::helper($name )
Retrieve helper object.

-    Mage:: register($key,$value,$graceful = false) )
Register a new variable.

-    Mage:: registry ($key )
Retrieve a value from registry by a key.
```

3.2. Calling methods to create class objects by config and through Mage class

- Mage::app()

```
/**
     * Get initialized application object.
     *
     * @param string $code
     * @param string $type
     * @param string|array $options
     * @return Mage_Core_Model_App
     */
public static function app($code = '', $type = 'store', $options =
array());
```

The Mage::app() function is used to bootstrap your Magento application (setting up configuration, autoloading and so on) and is useful when you want to access Magento models in your own custom script.

For example:

Mage::app()->getRequest->getParams(); Get the values sent from forms and params. Mage::app()->getRequest->getPost(); Get the values sent from forms by post method.

- Mage::getModel()

```
/**
     * Retrieve model object
     *
     * @link    Mage_Core_Model_Config::getModelInstance
     * @param   string $modelClass
     * @param   array $arguments
     * @return  Mage_Core_Model_Abstract
     */
public static function getModel($modelClass = '', $arguments = array());
```

This function is defined in file: app/Mage.php

For example:

Mage::getModel('lesson04/lesson04′);

If file config.xml of module is configured as follows:

```
<global>
    <models>
                <lesson04>
                    <class>Magestore_Lesson04_Model</class>

<resourceModel>lesson04_mysql4</resourceModel>
                </lesson04>
                <lesson04_mysql4>
                    <class>Magestore_Lesson04_Model_Mysql4</class>
                    <entities>
                        <lesson04>
                            <table>lesson04</table>
                        </lesson04>
                    </entities>
                </lesson04_mysql4>
    </models>
```

Function Mage::getModel('lesson04/lesson04′) will get config values of tag lesson04 in tag models (class Magestore_Lesson04_Model) and get config values of resourceModel of tag lesson04 (class Magestore_Lesson04_Model_Mysql4) table in database (lesson04).

- Mage::helper()

```
/**
     * Retrieve helper object
     *
     * @param string $name the helper name
     * @return Mage_Core_Helper_Abstract
     */
public static function helper($name);
```

This function is defined in file app/Mage.php.

Example 1: Mage::helper(‘lesson04’) ->test();

```
<helpers>
      <lesson04>
            <class>Magestore_Lesson04_Helper</class>
      </lesson04>
</helpers>
```

This piece of code will execute function test() in class defined in file config in (Magestore_Lesson04_Helper). This class is written in file app/code/local/Magestore/Lesson04/Helper/Data.php.

```
<?php
class Magestore_Lesson04_Helper_Data extends
Mage_Core_Helper_Abstract
{
    public function test(){
        return 'data helper';
        }
}
```

Example 2: Mage::helper(‘lesson04/newhelper’) ->test();

This piece of code will process the function test () in Magestore_Lesson04_Helper_Newhelper. This class is written in file app/code/local/Magestore/Lesson04/Helper/Newhelper.php.

```
<?php
class Magestore_Lesson04_Helper_Newhelper extends
Mage_Core_Helper_Abstract
{
    public function test(){
        return 'string';
        }
}
```

- Mage::getSingleton() :

```
/**
     * Retrieve model object singleton
     *
     * @param   string $modelClass
     * @param   array $arguments
     * @return  Mage_Core_Model_Abstract
     */
    public static function getSingleton($modelClass='', array
$arguments=array());
```

This function is defined in file: app/Mage.php

There are two ways to instantiate a model class:

```
Mage::getModel('groupname/classname');
Mage::getSingleton('groupname/classname');
```

The first form will give you a new class instance. The second form will give you a singleton class instance. The Magento abstraction allows you to create a singleton out of any Magento model class, but only if you stick to Magento instantiation methods. It means if you call:

```
Mage::getSingleton('groupname/classname');
```

will return that singleton instance (This is implemented with a registry pattern). However, nothing can prevent you from directly instantiating a new instance of the class with either

```
$o = Mage::getModel('groupname/classname');
$o = new Mage_Groupname_Model_Classname();
```

However, in many cases we can say:

Mage::getSingleton() get object that already exist (probably because of autoload mode of magento)

Mage::getModel() get a new one object and it was configed in config.xml.

- Mage::getResourceModel()

```
/**
     * Retrieve object of resource model
     *
     * @param   string $modelClass
     * @param   array $arguments
     * @return  Object
     */
```

```
    public static function getResourceModel($modelClass, $arguments = 
array());
```

This function is defined in file app/Mage.php.

ResourceModel is the layer that interacts with the Database objects and frames the queries. Models are classes that use the resource models to execute business logic. Normally, this is used for the models that communicate directly to the database. The path is required as a parameter to the location of the resource models Resource models normally are found in

/app/code/core/Mage/{Module}/Model/Mysql4 /

Example:

```
Mage::getResourceModel('catalog/product_collection')
```

This will get the model

/app/code/core/Mage/catalog/Model/Mysql4/product/collection.php

or:

```
Mage::getModel('catalog/product') ->getCollection();
```

III. SUMMARY

This is the end of lesson 04. This lesson enriches your Magento knowledge with:

- The objectives of file config.xml of each module;
- Common tags (xml tags) in file config.xml in each module configuration;
- How to name files, classes and actions in Magento;
- How to find path to files and classes based on corresponding class and file names;
- Common functions called through Mage class such as getModel() and helper(), getResourceModel(), as well as how to create them and add parameters to them.

So, you have now had a pretty good overview of the fundamentals of Magento. In future lessons, we will expand on these fundamentals.

Back to top

PART 02: MODULE DEVELOPMENT

Lesson 05: Module Setup and Upgrade

I. OVERVIEW

Magento is built on top of the Zend Framework that supports modularizing your application, and Magento goes the way of the module as part of its core design. In this lesson, I will explain how to use the *Module Creator* to help you create modules quickly.

1. Time: 2 hours

2. Content:

- Creating a new Magento Module by using Module Creator
- Creating Installer Script

II. CONTENT

1. Creating a new Magento Module by using Module Creator

The Module Creator allows you to fill in a few questions (Module Name, Namespace, Magento Root Dir, and some Design questions), then generates all the files necessary to get you started on your module. The coolest thing about the Module Creator is that you can create templates/skeletons/frameworks of modules and store them for later use. Module Creator is generally a time saver and recommended for noobs.

Step 1:

Use a browser to download the latest full released archive file from https://github.com/shell/Magento-Module-Creator. Then, extract Module Creator from the downloaded archive on your computer. Module Creator has folder Module Creator containing all of the Module Creator files created.

Step 2:

Copy Magento folder after extracting Module Creator from the downloaded archive to xampp\htdocs

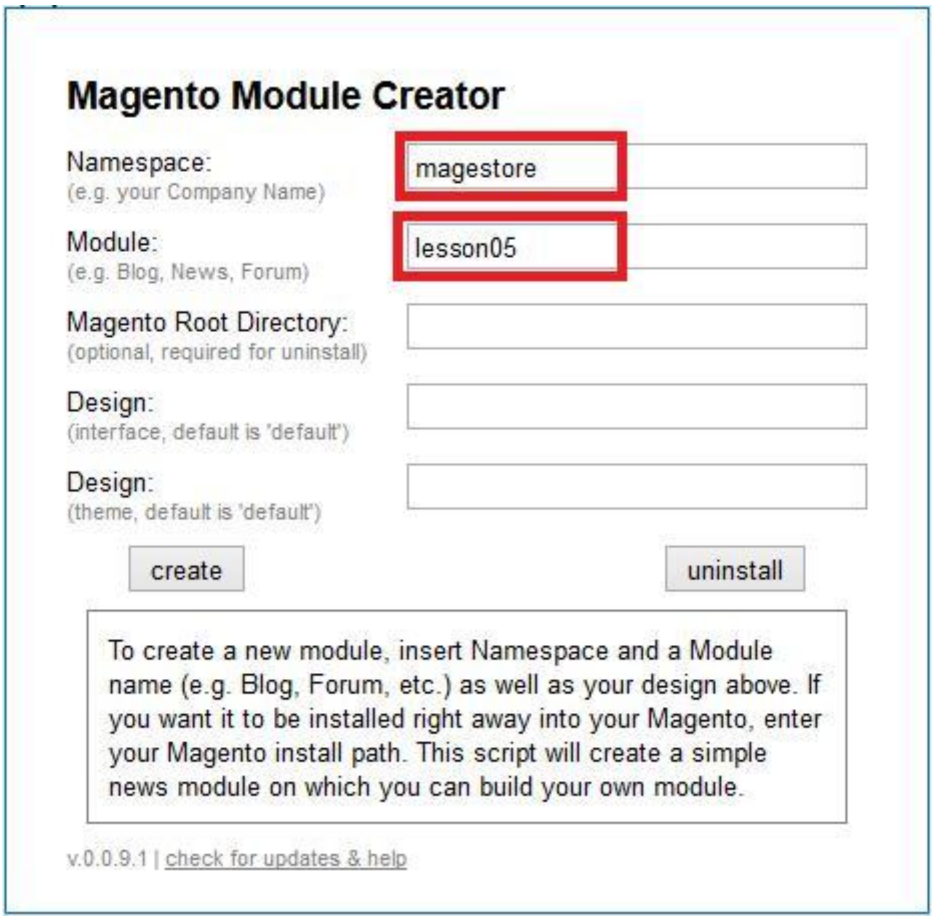
Step 3:

Use a browser, enter the URL: http://www.localhost.com/ModuleCreator to start creating new Magento Modules.

Fill information for Namespace and Module (Module name)

Example: Namespace: Magestore

Module: lesson05



Magento Module Creator

Namespace: (e.g. your Company Name)	magestore
Module: (e.g. Blog, News, Forum)	lesson05
Magento Root Directory: (optional, required for uninstall)	
Design: (interface, default is 'default')	
Design: (theme, default is 'default')	

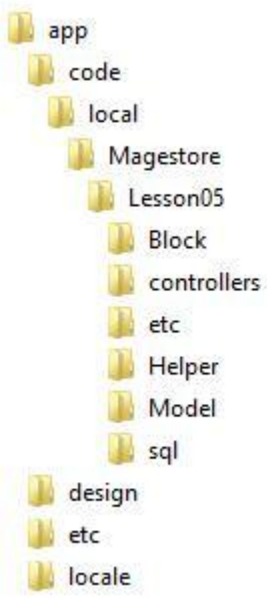
To create a new module, insert Namespace and a Module name (e.g. Blog, Forum, etc.) as well as your design above. If you want it to be installed right away into your Magento, enter your Magento install path. This script will create a simple news module on which you can build your own module.

v.0.0.9.1 | check for updates & help

Step 4:

Click on the Create button, all the source codes of new Magento Modules will be generated in *xampp\htdocs\ModuleCreator\new*.

After creating the new Magento Module by using Module Creator, you will see:



```
app
  code
    local
      Magestore
        Lesson05
          Block
          controllers
          etc
          Helper
          Model
          sql
  design
  etc
  locale
```

- **Declare Magestore_Lesson05 module** in file

  app/etc/modules/Magestore_Lesson05.xml

```
<?xml version="1.0"?>
<!--
/**
 * Magestore
 *
 * Online Magento Course
 *
 */
-->
<config>
    <modules>
        <!-- declare Magestore_Lesson05 module -->
        <Magestore_Lesson05>
            <active>true</active>
            <codePool>local</codePool>
        </Magestore_Lesson05>
    </modules>
</config>
```

- **Declare module's version information** in file

  app/code/local/Magestore/Lesson05/etc/config.xml

```
<?xml version="1.0"?>
<!--
/**
 * Magestore
 *
 * Online Magento Course
 *
 */
-->
<config>
    <modules>
        <!-- declare module's version information -->
        <Magestore_Lesson05>
            <!-- this version number will be used for database
upgrades -->
            <version>0.1.0</version>
        </Magestore_Lesson05>
    </modules>
    <global>
        <!-- declare model group for new module -->
        <models>
            <!-- model group alias to be used in Mage::getModel() --
>
            <lesson05>
                <!-- base class name for the model group -->
                <class>Magestore_Lesson05_Model</class>


           <!-- declare model lesson05 work with lesson05 table -->
           <entities>
           <lesson05>
                   <table>lesson05</table>
               </lesson05>
           </entities>
      </lesson05>
 </models>

        <!-- declare resource setup for new module -->
        <resources>
            <!-- resource identifier -->
            <lesson05_setup>
                <!-- specify that this resource is a setup resource
and used for upgrades -->
                <setup>
                    <!-- which module to look for install/upgrade
files in -->
                    <module>Magestore_Lesson05</module>
                </setup>
                <!-- specify database connection for this resource -
->
                <connection>
                    <!-- do not create new connection, use
predefined core setup connection -->
                    <use>core_setup</use>
                </connection>
            </lesson05_setup>
        </resources>
    </global>
</config>
```

- **Lesson05 model**: *app/code/local/Magestore/Lesson05/Model/Lesson05.php*

```
<?php
/**
 * Magestore
 *
 * Online Magento Course
 *
 */
/**
 * Lesson05 Model
 *
 * @category    Magestore
 * @package     Magestore_Lesson05
 * @author      Magestore Developer
 */
class Magestore_Lesson05_Model_Lesson05 extends Mage_Core_Model_Abstract
{
    public function _construct()
    {
        parent::_construct();
        $this->_init('lesson05/lesson05');
    }
}
```

- **Lesson05 resource model**:

  app/code/local/Magestore/Lesson05/Model/Mysql4/Lesson05.php

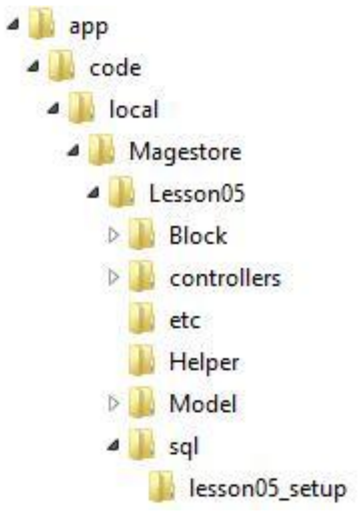
```
<?php
/**
 * Magestore
 *
 * Online Magento Course
 *
 */

/**
 * Lesson05 Resource Model
 *
 * @category    Magestore
 * @package     Magestore_Lesson05
 * @author      Magestore Developer
 */
class Magestore_Lesson05_Model_Mysql4_Lesson05 extends
Mage_Core_Model_Mysql4_Abstract
{
    public function _construct()
    {
      /* lesson05_id is primary of lesson05 table */
        $this->_init('lesson05/lesson05', 'lesson05_id');
    }
}
```

- **Lesson05 Resource Collection Model**:

  app/codel/local/Magestore/Lesson05/Model/Mysql4/Lesson05/Collection.php

```
<?php
/**
 * Magestore
 *
 * Online Magento Course
 *
 */

/**
 * Lesson05 Resource Collection Model
 *
 * @category     Magestore
 * @package      Magestore_Lesson05
 * @author       Magestore Developer
 */
class Magestore_Lesson05_Model_Mysql4_Lesson05_Collection extends
Mage_Core_Model_Mysql4_Collection_Abstract
{
    public function _construct()
    {
        parent::_construct();
        $this->_init('lesson05/lesson05');
    }
}
```

- All file sql to install database of module in*app/code/local/Magestore/Lesson05/sql/lesson05_setup*



2. Creating your Installer Magento Script

Next, you will want to create your installer script. This is the script that will contain any CREATE TABLE or other SQL code that needs to be run to initialize our module. This section will help you create the database for module, update database for module, etc.

First, look at your config.xml file.

Example: *app/code/local/Magestore/Lesson05/etc/config.xml*

```
<modules>
    <Magestore_Lesson05>
        <version>0.1.0</version>
    </Magestore_Lesson05>
</modules>
```

and

```
<global>
       <resources>
            <lesson05_setup>
                <setup>
                    <module>Magestore_Lesson05</module>
                </setup>
                <connection>
                    <use>core_setup</use>
                </connection>
            </lesson05_setup>
            <lesson05_write>
                <connection>
                    <use>core_write</use>
                </connection>
            </lesson05_write>
            <lesson05_read>
                <connection>
                    <use>core_read</use>
                </connection>
            </lesson05_read>
        </resources>
    </global>
```

This section is required in all config.xml files, and identifies the module as well as its version number. Your installer Magento script's name will be based on this version number. The following example assumes the current version of your module is 0.1.0. Define the resource setup for module and connect to database with read/write database.

Create the following file at the following location:

File: *app/code/local/Magestore/Lesson05/sql/lesson05_setup/mysql4-install-0.1.0.php*

Add the following code to your setup script.

```
$installer = $this;

$installer->startSetup();
```

```
/**
 * create lesson05 table
 */
$installer->run("
      DROP TABLE IF EXISTS {$this->getTable('lesson05')};
      CREATE TABLE {$this->getTable('lesson05')} (
        `lesson05_id` int(11) unsigned NOT NULL auto_increment,
        `title` varchar(255) NOT NULL default '',
        `content` text NOT NULL default '',
        `status` smallint(6) NOT NULL default '0',
        PRIMARY KEY (`lesson05_id`)
      ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

      INSERT INTO `{$this->getTable('lesson05')}` VALUES (1,'Magento
Course','Hello, I am Michael from Magestore.com',1);
      ");
$installer->endSetup();
```

Also, you can use some functions Magento supports to work with database:

- **addColumn** ($name, $type, $size = null, $options = array(), $comment = null)
  - $name: the name of a field (column)
  - $type: the type of data. For instance: TYPE_BOOLEAN, TYPE_SMALLINT, TYPE_INTEGER…
  - $size: the size of a field (such as 0, 255 and 2M)
  - $option: identity (true/false), nullable (true/false), primary (true/false), default (value)
  - $comment: add comments to the field
- **addForeignKey** ($fkName, $column, $refTable, $refColumn, $onDelete = null, $onUpdate = null)
  - $fkName: the name of the foreign key
  - $column: the name of the field (column) set into the foreign key
  - $ refTable: the name of the reference table
  - $ refColumn: the name of the column table

  - $onDelete: identify an action needs to be implemented when the data of the reference table is deleted (ACTION_CASCADE, ACTION_SET_NULL, ACTION_NO_ACTION,ACTION_RESTRICT, CTION_SET_DEFAULT)
  - $onUpdate: : identify an action needs to be implemented when the data of the reference table is updated (ACTION_CASCADE,ACTION_SET_NULL, ACTION_NO_ACTION, ACTION_RESTRICT, ACTION_SET_DEFAULT)

- **addIndex** ($indexName, $fields, $options = array())
  - $indexName: the name of index
  - $fields: the name/ array of field which is set into index (array or string)

Example:

```
$installer = $this;

/**
 * create lesson05 table
 */

$table = $installer->getConnection()
    ->newTable($installer->getTable('lesson05'))
    ->addColumn('lesson05_id', Varien_Db_Ddl_Table::TYPE_INT, 11, array(
        'identity'  => true,
        'nullable'  => false,
        'primary'   => true,
        ), 'Lesson05 ID')
    ->addColumn('title', Varien_Db_Ddl_Table::TYPE_VARCHAR, 255, array(
        'nullable'  => false,
        ), 'Lesson05 Title')
    ->addColumn('content', Varien_Db_Ddl_Table::TYPE_TEXT, '2M', array(
        ), 'Lesson05 Content')
    ->addColumn('status', Varien_Db_Ddl_Table::TYPE_SMALLINT, 6, array(
        'nullable'  => false,
        'default'   => '0',
        ), 'Status')
    ->setComment('Lesson05 Table');
$installer->getConnection()->createTable($table);
```

The lesson05_setup portion of the path should match the tag you created in your config.xml file (<lesson05_setup />). The 0.1.0 portion of the filename should match the starting version of your module.

Resource Versions

Magento Setup Resources allow you to drop your installation scripts (and upgrade scripts) onto the server, and have the system automatically run them. This allows you to have all your database migrations scripts stored in the system in a consistent format.

Use your favorite database client and look at the core_setup table:

```
mysql> select * from core_resource;
    +-------------------------+------------+--------------+
    | code                    | version    | data_version |
    +-------------------------+------------+--------------+
    | adminnotification_setup | 1.6.0.0    | 1.6.0.0      |
    | admin_setup             | 1.6.1.0    | 1.6.1.0      |
    | api2_setup              | 1.0.0.0    | 1.0.0.0      |
    | api_setup               | 1.6.0.0    | 1.6.0.0      |
    | backup_setup            | 1.6.0.0    | 1.6.0.0      |
    | bundle_setup            | 1.6.0.0.1  | 1.6.0.0.1    |
    | captcha_setup           | 1.7.0.0.0  | 1.7.0.0.0    |
    | catalogindex_setup      | 1.6.0.0    | 1.6.0.0      |
    | cataloginventory_setup  | 1.6.0.0.2  | 1.6.0.0.2    |
    | catalogrule_setup       | 1.6.0.3    | 1.6.0.3      |
    | catalogsearch_setup     | 1.6.0.0    | 1.6.0.0      |
    | catalog_setup           | 1.6.0.0.14 | 1.6.0.0.14   |
    | checkout_setup          | 1.6.0.0    | 1.6.0.0      |
    | cms_setup               | 1.6.0.0.1  | 1.6.0.0.1    |
    | compiler_setup          | 1.6.0.0    | 1.6.0.0      |
    | contacts_setup          | 1.6.0.0    | 1.6.0.0      |
    | core_setup              | 1.6.0.2    | 1.6.0.2      |
    | cron_setup              | 1.6.0.0    | 1.6.0.0      |
    | customer_setup          | 1.6.2.0.1  | 1.6.2.0.1    |
    | dataflow_setup          | 1.6.0.0    | 1.6.0.0      |
    | directory_setup         | 1.6.0.1    | 1.6.0.1      |
    | downloadable_setup      | 1.6.0.0.2  | 1.6.0.0.2    |
    | eav_setup               | 1.6.0.0    | 1.6.0.0      |
    | giftmessage_setup       | 1.6.0.0    | 1.6.0.0      |
    | googleanalytics_setup   | 0.1.0      | 0.1.0        |
    | googlecheckout_setup    | 1.6.0.1    | 1.6.0.1      |
    | importexport_setup      | 1.6.0.2    | 1.6.0.2      |
    | index_setup             | 1.6.0.0    | 1.6.0.0      |
    | lesson05_setup          | 0.1.0      | 0.1.0        |
    | log_setup               | 1.6.0.0    | 1.6.0.0      |
    | moneybookers_setup      | 1.6.0.0    | 1.6.0.0      |
    | newsletter_setup        | 1.6.0.1    | 1.6.0.1      |
    | oauth_setup             | 1.0.0.0    | 1.0.0.0      |
    | paygate_setup           | 1.6.0.0    | 1.6.0.0      |
    | payment_setup           | 1.6.0.0    | 1.6.0.0      |
    | paypaluk_setup          | 1.6.0.0    | 1.6.0.0      |
    | paypal_setup            | 1.6.0.2    | 1.6.0.2      |
    | persistent_setup        | 1.0.0.0    | 1.0.0.0      |
    | poll_setup              | 1.6.0.0    | 1.6.0.0      |
    | productalert_setup      | 1.6.0.0    | 1.6.0.0      |
    | rating_setup            | 1.6.0.0    | 1.6.0.0      |
    | reports_setup           | 1.6.0.0.1  | 1.6.0.0.1    |
    | review_setup            | 1.6.0.0    | 1.6.0.0      |
    | salesrule_setup         | 1.6.0.3    | 1.6.0.3      |
    | sales_setup             | 1.6.0.7    | 1.6.0.7      |
    | sendfriend_setup        | 1.6.0.0    | 1.6.0.0      |
    | shipping_setup          | 1.6.0.0    | 1.6.0.0      |
    | sitemap_setup           | 1.6.0.0    | 1.6.0.0      |
    | tag_setup               | 1.6.0.0    | 1.6.0.0      |
    | tax_setup               | 1.6.0.3    | 1.6.0.3      |
    | usa_setup               | 1.6.0.1    | 1.6.0.1      |
    | weee_setup              | 1.6.0.0    | 1.6.0.0      |
    | widget_setup            | 1.6.0.0    | 1.6.0.0      |
    | wishlist_setup          | 1.6.0.0    | 1.6.0.0      |
    | xmlconnect_setup        | 1.6.0.0    | 1.6.0.0      |
    +-------------------------+------------+--------------+
    55 rows in set (0.00 sec)
```

This table contains a list of all installed modules, along with the installed version numbers. You can see our module near the end

```
| lesson05_setup             | 0.1.0      | 0.1.0        |
```

You cannot rerun your script on the second, and on all successive page loads. The lesson05_setup has already installed, so it will not be updated. If you want to rerun your installer script (it is useful when you are developing), delete the row of your module from this table, and add the SQL to create our table. So first, run the following SQL.

Anatomy of a Setup Script

So, go over the script line-by-line

```
$installer = $this;
```

Each installer script is run from the context of a Setup Resource class, the class you created above. That means any reference to $this from within the script will be a reference to an object instantiated from this class. While not necessary, most setup scripts in the core modules will use alias $this for a variable called installer, which is what we have done here. While not necessary, it is the convention and it is always best to follow the convention unless you have a good reason for breaking it.

Next, you see our queries bookended by the following two method calls.

```
$installer->startSetup();
//...
$installer->endSetup();
```

If you look at the Mage_Core_Model_Resource_Setup class in app/code/core/Mage/Core/Model/Resource/Setup.php (which your setup class inherits from) you can see that these methods do some basic SQL setup

```
public function startSetup()
    {
        $this->getConnection()->startSetup()
        return $this;
    }

    public function endSetup()
    {
        $this->getConnection()->endSetup();
        return $this;
    }
```

Look can into Varien_Db_Adapter_Pdo_Mysql to find the real SQL setup executed for MySQL connections in the startSetup() and endSetup() methods.

Finally, there is a call to the run method

```
$installer->run(...);
```

which accepts a string containing the SQL needed to setup your database table(s). You may specify any number of queries, separated by a semi-colon. You also probably noticed the following:

```
$this->getTable('lesson05')
```

The getTable method allows you to pass in a Magento Model URI and get its table name. If unnecessary, using this method ensures that your script will continue to run, even if someone changes the name of the table in the config file. The Mage_Core_Model_Resource_Setup class contains many useful helper methods like this. The best way to become familiar with everything that is possible is to study the installer scripts used by the core Magento modules.

Module Upgrade

So, that how you create a script will set up your initial database tables, but what if you want to alter the structure of an existing module? Magento Setup Resources support a simple versioning scheme that will let you automatically run scripts to upgrade modules.

Once Magento runs an *installer* script for a module, it will never run another installer for that module again (short of manually deleting the reference in the core_resource table). Instead, you will need to create an upgrade script. Upgrade scripts are very similar to installer scripts, with a few key differences.

To get started, we will create a script at the following location, with the following contents:

File: app/code/local/Magestore/Lesson05/sql/lesson05_setup/upgrade-0.1.0-0.2.0.php:

```
DROP TABLE IF EXISTS {$this->getTable('lesson05_upgrade')};

CREATE TABLE {$this->getTable('lesson05_upgrade'')} (
  `lesson05_upgrade_id` int(11) unsigned NOT NULL auto_increment,
  `title` varchar(255) NOT NULL default '',
  `content` text NOT NULL default '',
  `status` smallint(6) NOT NULL default '0',
  PRIMARY KEY (`lesson05_upgrade_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Upgrade scripts are placed in the same folder as your installer script, but they are named slightly differently. Firstly, the file name contains the word upgrade. Secondly, you will notice there are **two** version numbers, separated by a "-". The first (0.1.0) is the module version that we are upgrading **from**. The second (0.2.0) is the module version we are upgrading **to**.

Notice: If you want to update database from version 0.1.0 to version 0.2.0, the naming file for SQL file is:

"upgrade" + "-" + "0.1.0" + "-" + "0.2.0" = upgrade-0.1.0-0.2.0

If we clear our Magento cache and reload a page, our script will not run. We need to update the the version number in our module's config.xml file to trigger the upgrade

```
<modules>
    <Magestore_Lesson05>
        <version>0.2.0</version>
    </Magestore_Lesson05>
</modules>
```

With the new version number in place, we'll need to clear our Magento cache and load any page in our Magento site. You should now see output from your upgrade script.

By the way, we also could have names our upgrade script **mysql4-**upgrade-0.1.0-0.2.0.php. This would indicate our upgrade would contain MySQL specific SQL.

```
mysql> select * from core_resource where code = lesson05_setup';

+--------------+---------+--------------+

| code         | version | data_version |

+--------------+---------+--------------+

|lesson05_setup| 0.2.0   | 0.2.0        |

+--------------+---------+--------------+

1 row in set (0.00 sec)
```

Here's what happened:

- The lesson05_setup resource was at version 0.1.0
- We upgraded our module to version 0.2.0
- Magento saw the upgraded module, and there were two upgrade scripts to run, 0.1.0 to 0.2.0
- Magento queued up both scripts to run
- Magento ran the 0.1.0 to 0.2.0 script
- The lesson05_setup resource is now at version 0.2.0

The correct way to achieve what we wanted is to name our scripts as follows:

```
upgrade-0.1.0-0.2.0.php #This goes 0.1.0 to 0.2.0
```

III. SUMMARY

After learning this lesson, you can answer two questions:

- How to create a new Magento Module by using Module Creator.
- How to create Installer Script:

- Create database for module
- Upgrade database for module

Back to top

Lesson 06: Creating a Menu in Backend

I. OVERVIEW

1. Time: 3 hours

2. Content

- Configure the adminhtml router
- Add Magento menu from file adminhtml.xml and configure ACL for menus
- Add controllers for each Magento menu

II. CONTENT

1. Creating a Magento menu in backend

1.1. Creating a Magento menu from file adminhtml.xml

- Menu Item

```
<?xml version="1.0"?>
<config>
    <menu>
        <[menu identify] module="[module name] " translate="title">
            <title>[title]</title>
            <sort_order>[number sort order]</sort_order>
            <children>
                <[menu child identify]  module="[module name] "
translate="title">
                    <title>[title 2]</title>
                    <sort_order>[ number sort order] </sort_order>
                        <action>
                [module adminhtml router]/adminhtml_[controller]
            </action>
                </[menu child identify]  >
                <settings module="[module name] " translate="title">
                    <title>[title 3] </title>
                    <sort_order>[number sort order]</</sort_order>
                    <action>[module/controller/action] </action>
                </settings>
            </children>
        </[menu identify]>
    </menu>
```

- Meaning of tags
    - *<title>* tag: Assigning a label to the menu, displaying on horizontal navigation menu bar.

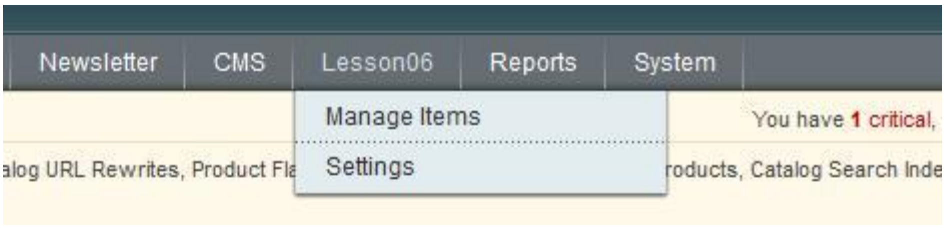
- ○ *<sort_order>* tag: Assigning sort value to each menu, by left-to-right, top-to-bottom order; defined by increasing value of sort_order value.
- ○ *<action>* tag: Defining the controller/action that processing this menu; similar to assigning links to the menu.

- Menu displayed in backend

For example:

File app\code\local\Magestore\Lesson06\etc\adminhtml.xml

```
<?xml version="1.0"?>
<config>
    <menu>
        <lesson06 module="lesson06" translate="title">
            <title>Lesson06</title>
            <sort_order>71</sort_order>
            <children>
                <lesson06 module="lesson06" translate="title">
                    <title>Manage Items</title>
                    <sort_order>0</sort_order>
                        <action>
                lesson06admin/adminhtml_lesson06
            </action>
                </lesson06>
                <settings module="lesson06" translate="title">
                    <title>Settings</title>
                    <sort_order>1000</sort_order>

<action>adminhtml/system_config/edit/section/lesson06</action>
                </settings>
            </children>
        </lesson06>
    </menu>
```
Result:



1.2. Creating ACL to help setting permissions to Magento menus

- Defining ACL in file adminhtml.xml

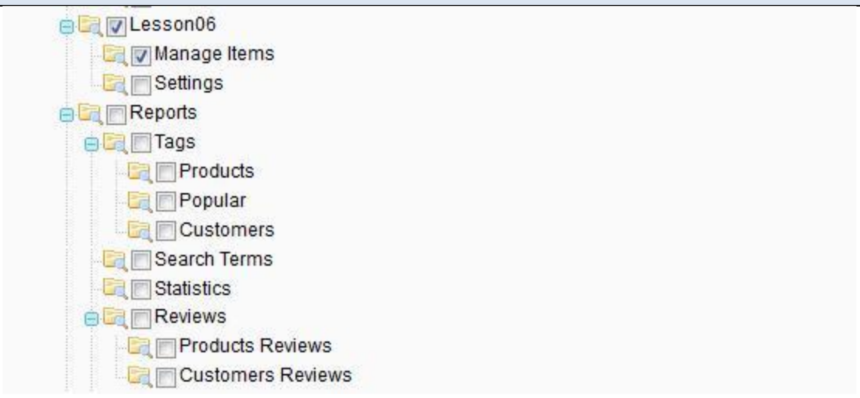
For example:

```
<acl>
        <resources>
            <all>
                <title>Allow Everything</title>
            </all>
            <admin>
                <children>
                    <system>
                        <children>
                            <config>
                                <children>
                                    <lesson06 module="lesson06"
translate="title">
                                        <title>Lesson06</title>
                                        <sort_order>71</sort_order>
                                    </lesson06>
                                </children>
                            </config>
                        </children>
                    </system>
                    <lesson06 module="lesson06" translate="title">
                        <title>Lesson06</title>
                        <sort_order>71</sort_order>
                        <children>
                            <lesson06 module="lesson06" translate="title">
                                <title>Manage Items</title>
                                <sort_order>0</sort_order>
                            </lesson06>
                            <settings module="lesson06" translate="title">
                                <title>Settings</title>
                                <sort_order>1000</sort_order>
                            </settings>
                        </children>
                    </lesson06>
                </children>
            </admin>
        </resources>
    </acl>
```

- Creating roles to set permissions for users in backend

You can create roles to the new Magento menus by accessing backend following this path:

```
System -> Permissions -> Roles -> Add New Role
```



- ☑ Lesson06
  - ☑ Manage Items
  - ☐ Settings
- ☐ Reports
  - ☐ Tags
    - ☐ Products
    - ☐ Popular
    - ☐ Customers
  - ☐ Search Terms
  - ☐ Statistics
  - ☐ Reviews
    - ☐ Products Reviews
    - ☐ Customers Reviews

- Applying ACL to Magento

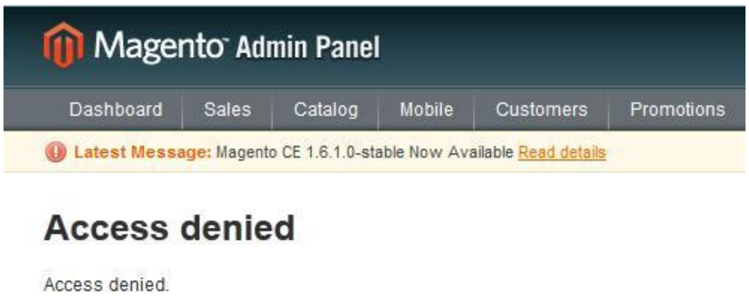
File: app\code\local\Magestore\Lesson06\controllers\Adminhtml\Lesson06Controller.php

```
protected function _isAllowed()
```

```
{
   return Mage::getSingleton('admin/session')->isAllowed('lesson06');
}
```

- Effects of permission settings to corresponding Admin User

If you do not add roles to a user (as above), there is a denied error message when that user accesses a Magento menu:



Latest Message: Magento CE 1.6.1.0-stable Now Available Read details

Access denied

Access denied.

2. Creating a Magento controller action in backend

2.1. Declaring a router in file config.xml

File app\code\local\Magestore\Lesson06\etc\config.xml

Overall configuration:

```
<admin>
     <routers>
          <adminhtml>
                  <args>
                          <modules>
                                  <[Namespace]_[Modulename]_[Adminhtml]
>
[Namespace]_[Modulename]_Adminhtml</[Namespace]_[Modulename]_[Adminht
ml]>
                          </modules>
                  </args>
          </adminhtml>
    </routers>
</admin>
```

Example:

```
<admin>
        <routers>
            <lesson06admin>
                <use>admin</use>
                <args>
                    <module>Magestore_Lesson06</module>
                    <frontName>lesson06admin</frontName>
                </args>
            </lesson06admin>
        </routers>
    </admin>
```

- *<admin>* and *<routers>* are the parent nodes. <admin> can be either
    - *<admin>*: for admin routers or
    - *<frontend>*: for frontend routers
- *<lesson06admin>*: It is the name of your module in lowercase. You can indicate a name that is already in use (such as adminhtml) only if you want to extend the configuration of that route with a child entry in <modules> node. Other settings will not be rewritten or may even cause errors.
- *<args>*: Node can contain three different types of arguments.
- *<module>*: It contains the full name of your module including package (Packagename_Modulename). The system will then look for controllers in the controller directory of the module you indicated.
- *<frontName>*: It sets the router name as it will be included in the URL: http://yourwebsite.com/index.php /frontname/controller/action like

  http://yourwebsite.com/index.php/lesson06admin/

  adminhtml_testbymagestore/index/key/69a015b1c7b4151122e352de306e1d0c/

2.2. Link structure in Magento

- In backend:

Example: http://yourwebsite.com/index.php/lesson06admin/adminhtml_lesson06/view

in which:

lesson06admin: the adminhtml router

adminhtml_lesson06: the controller name requested in backend (in folder adminhtml) *view*: the action in controller lesson06 requested.

- In frontend:

Example: http://yourwebsite.com/index.php /lesson06/ lesson06/view

lesson06: the frontend router

lesson06: the controller requested in frontend

view: the action in the controller lesson06 requested.

2.3. Controller for each menu

- Adding controllers for each menu

Example:

File app\code\local\Magestore\Lesson06\etc\adminhtml.xml

```
<?xml version="1.0"?>
<config>
    <menu>
        <lesson06 module="lesson06" translate="title">
            <title>Lesson06</title>
            <sort_order>71</sort_order>
            <children>
                <lesson06 module="lesson06" translate="title">
                    <title>Manage Items</title>
                    <sort_order>0</sort_order>
                <action>
                    lesson06admin/adminhtml_lesson06
            </action>
                </lesson06>
            . . .
   </menu>
</config>
```

File controller:

app\code\local\Magestore\Lesson06\controllers\Adminhtml\Lesson06Controller.php

```
<?php
class Magestore_Lesson06_Adminhtml_Lesson06Controller extends
Mage_Adminhtml_Controller_Action
{
    protected function _initAction()
    {
        $this->loadLayout()
            ->_setActiveMenu('lesson06/lesson06')
            ->_addBreadcrumb(
                Mage::helper('adminhtml')->__('Items Manager'),
                Mage::helper('adminhtml')->__('Item Manager')
            );
        return $this;
    }

    public function indexAction()
    {
        $this->_initAction()
            ->renderLayout();
    }
}
```

Example 2:

File app\code\local\Magestore\Lesson06\etc\adminhtml.xml

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

```
         <lesson06 module="lesson06" translate="title">
             <title>Lesson06</title>
             <sort_order>71</sort_order>
             <children>
                 <lesson06 module="lesson06" translate="title">
                     <title>Test Action Menu</title>
                     <sort_order>1</sort_order>
                 <action>
                     lesson06admin/adminhtml_lesson06/test
             </action>
                 </lesson06>
```

- URL of action menu:

```
http://yourwebsite.com/index.php/lesson06admin/adminhtml_lesson06/test/key/
..../
```

File controller:

app\code\local\Magestore\Lesson06\controllers\Adminhtml\Lesson06Controller.php

```
<?php
class Magestore_Lesson06_Adminhtml_Lesson06Controller extends
Mage_Adminhtml_Controller_Action
{
    public function testAction(){
        $block = $this->getLayout()->
createBlock('adminhtml/template')->
setTemplate('lesson06/test/rendertesttemplate.phtml');
        echo $block->toHtml();
    }
protected function _isAllowed()
    {
        return Mage::getSingleton('admin/session')-
>isAllowed('lesson06');
    }
```

III. SUMMARY

After finishing Lesson 6, you are supposed to know:

- Where adminhtml router of module Magento is and how to configure it
- Common xml tags of file adminhtml.xml in admin configuration for each module
- How to configure ACL (admin and user permission settings) for horizontal menu in Magento backend
- Writing controller and action of menu in admin page.

Back to top

Lesson 07: Creating a Grid Management Page in Backend

I. OVERVIEW

1. Time: 2-3 hours

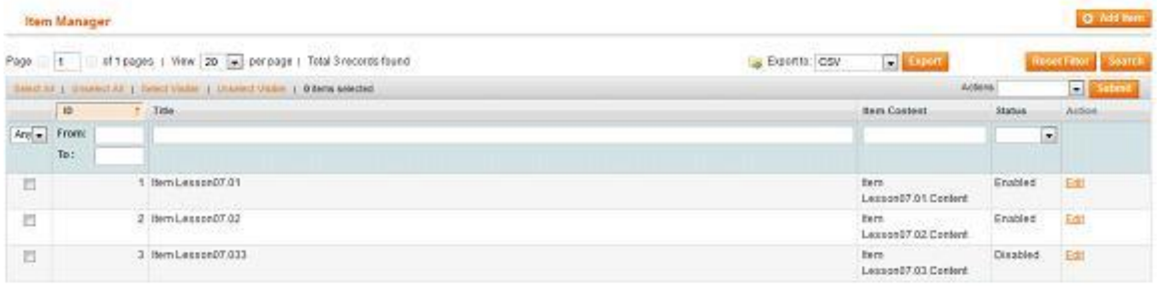
2. Content:

- Creating controller/action for Grid Page
- Adding layout for Grid Page
- Creating Block Grid (with database collections and columns of grid)

II. CONTENT

*** Grid in Magento Backend**

Grid can display data in the form of a table.


Item Manager
Add Item
Page 1 of 1 pages | View 20 per page | Total 3 records found
Exports: CSV
Export
Reset Filter
Search
Select All | Unselect All | Select Visible | Unselect Visible | 0 items selected
Actions
Submit
Any
From:
To:

	ID	Title	Item Content	Status	Action
	1	Item Lesson07.01	Item Lesson07.01 Content	Enabled	Edit
	2	Item Lesson07.02	Item Lesson07.02 Content	Enabled	Edit
	3	Item Lesson07.033	Item Lesson07.03 Content	Disabled	Edit

Some basic concepts for grid:

- **Grid container**: It extends from class Mage_Adminhtml_Block_Widget_Grid_Container with the default template *widget/grid/container.phtml*. Grid container includes grid's header (title and buttons) and grid's content. Grid Container represents the skeleton of a grid page. It includes button, data grid and so on.
- **Grid content**: It extends from class Mage_Adminhtml_Block_Widget_Grid and has default template *widget/grid.phtml*. Grid content includes a data grid, paging grid, filter data, and massaction for grid. Grid has an important method called addColumn.
- **Massaction**: It bases on Mage_Adminhtml_Block_Widget_Grid_Massaction with the default template *widget/grid/massaction.phtml*. Massaction used to operate list of data in grid.

- **Serializer**: It works with grid data (by javascript) and transform it to serial.

1. Creating controller/action for Magento grid page in backend

Note: Review lesson 06 for instruction on how to create link to grid by using menu.

File:app/code/local/Magestore/Lesson07/controllers/Adminhtml/Lesson07Controller.php

```
class Magestore_Lesson07_Adminhtml_Lesson07Controller extends
Mage_Adminhtml_Controller_Action
{
    protected function _initAction()
    {
        $this->loadLayout()
            ->_setActiveMenu('lesson07/lesson07')
            ->_addBreadcrumb(
                Mage::helper('adminhtml')->__('Items Manager'),
                Mage::helper('adminhtml')->__('Item Manager')
            );
        return $this;
    }
    public function indexAction()
    {
        $this->_initAction()
            ->renderLayout();
    }
```

$this->loadLayout(): Get configured data in file layout (app/design/frontend/default/default/layout/lesson07.xml) including block for controller, reference block , templates for controller, JS, CSS and so on.

```
/**

     *in class Mage_Adminhtml_Controller_Action extends
     * Mage_Core_Controller_Varien_Action
     * in file app/code/core/Mage/Adminhtml/Controller/Action.php
     */
    public function loadLayout($ids=null, $generateBlocks=true,
$generateXml=true);
```

Mage::helper('adminhtml')->__('Item Manager') is similar to echo but supports translating into other languages.

- $this ->_setActiveMenu('lesson07/lesson07′): define active menu item in menu block
- $this->_addBreadcrumb(): define active menu item in menu block.
- *$this->renderLayout():* after loading layout, information will be rendered into html and displayed on screen.

```
/**
     * abstract class Mage_Core_Controller_Varien_Action
     * in file app/code/core/Mage/Core/Controller/Varien/Action.php
     * Rendering layout
     *
     * @param   string $output
     * @return  Mage_Core_Controller_Varien_Action
     */
    public function renderLayout($output='');
```

2. Creating layout for Magento Grid page

File: app/design/adminhtml/default/default/layout/Lesson07.xml

```
<?xml version="1.0"?>
<layout version="0.1.0">
    <lesson07admin_adminhtml_lesson07_index>
        <reference name="content">
            <block type="lesson07/adminhtml_lesson07" name="lesson07"
/>
        </reference>
    </lesson07admin_adminhtml_lesson07_index>
</layout>
```

3. Creating a Magento grid block (with data collection and grid columns)

3.1. Widget grid Container

File: app/code/local/Magestore/Lesson07/Block/Adminhtml/Lesson07.php

Grid Container represents the skeleton of a grid page. It includes button, data grid and so on. Below is the code for you to define a Grid Container.

```
class Magestore_Lesson07_Block_Adminhtml_Lesson07 extends
Mage_Adminhtml_Block_Widget_Grid_Container
{
    publ
        $this->_controller = 'adminhtml_lesson07';
        $this->_blockGroup = 'lesson07';
        $this->_headerText = Mage::helper('lesson07')->__('Item
Manager');
        $this->_addButtonLabel = Mage::helper('lesson07')->__('Add
Item');
        parent::__construct();
    }
}
ic function __construct()
    {
```

- $this->_controller: declares controller in charge of receiving and executing data from forms in the grid
- *$this->_blockGroup*: declares block (group block) file for the grid
- *$this->_headerText*: declares the label of header for the grid
- *$this->_addButtonLabel*: declares the label for the 'add' button

The default values of these variables are created in class Mage_Adminhtml_Block_Widget_Container in file app/code/core/Mage/Adminhtml/Block/Widget/Container.php

3.2. Grid content

File: app/code/local/Magestore/Lesson07/Block/Adminhtml/Lesson07/Grid.php

- _construct()

```
public function __construct(){
        parent::__construct();
        $this->setId('lesson07Grid');
        $this->setDefaultSort('lesson07_id');
        $this->setDefaultDir('ASC');
        $this->setSaveParametersInSession(true);
    }
```

$this->setId('lesson07Grid'): declares ID for the div tag containing grid and prefix for classes of other html elements.

$this->setDefaultSort('lesson07_id'): sets the default sort value when grid is loaded.

$this->setDefaultDir('ASC'): sets the default sort value in grid based on ASC.

- _prepareCollection(): Prepares the data for presenting grid's data

```
protected function _prepareCollection(){
        $collection = Mage::getModel('lesson07/lesson07') ->
getCollection();
        $this->setCollection($collection);
        return parent::_prepareCollection();
    }
```

Mage::getModel('lesson07/lesson07'): gets Model from config.

Mage::getModel('lesson07/lesson07') -> getCollection(): gets collection data of model 'lesson07/lesson07'.

$this->setCollection($collection): sets collection for the whole grid.

- _prepareColumns(): Adds columns to grid and define columns to be displayed

```
protected function _prepareColumns()
    {
        $this->addColumn('lesson07_id', array(
            'header'    => Mage::helper('lesson07')->__('ID'),
            'align'     =>'right',
            'width'     => '50px',
            'index'     => 'lesson07_id',
            'type'      => 'number',
        ));

        $this->addColumn('title', array(
            'header'     => Mage::helper('lesson07')->__('Title'),
            'align'      =>'left',
            'index'      => 'title',
        ));

        $this->addColumn('content', array(
            'header'     => Mage::helper('lesson07')->__('Item Content'),
            'width'      => '150px',
            'index'      => 'content',
        ));

        $this->addColumn('status', array(
            'header'     => Mage::helper('lesson07')->__('Status'),
            'align'      => 'left',
            'width'      => '80px',
            'index'      => 'status',
            'type'         => 'options',
            'options'      => array(
                1 => 'Enabled',
                2 => 'Disabled',
            ),
        ));
}
```

Here we use the function $this->addColumn(): Whenever this function is used, a new column will be created for the grid table. If the parameter "index" is used, one field in the database will go with one new column. The parameters of this function are:

- header: assign label of column.
- width: set width for column.
- type: set type for column. Types are keywords in Magento grid. You can refer more to lesson 8.
- url: assign module/controller/action for column.
- filter: enable or disable filter/search in current column.
- sortable: enable or disable sort in current column.
- index: set the value of corresponding column with corresponding data field in database.
- Index: sets the value of the column corresponding with the data in the database.
- align: is the align attribute in css.

- _prepareMassaction(): This will be explained in detail in lesson 8.
- getRowUrl($row)

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

Add a URL to each row in grid to be redirected to the URL page when clicking on that row.

4. Creating Magento Ajax Grid

Using Ajax for grid helps sort and search right on current pages without loading to a new page.

In order to ***create an Ajax grid*** for the above grid, we can follow these steps:

- *Creating controller/action*

File: app/code/local/Magestore/Lesson07/controllers/Adminhtml/Lesson07Controller.php adds function gridAction().

```
public function gridAction(){
        $this->loadLayout();
        $this->renderLayout();
    }
```

- *Creating layout for ajax grid*

```
<lesson07admin_adminhtml_lesson07_grid>
        <block type="core/text_list" name="root">
            <block type="lesson07/adminhtml_lesson07_grid"
name="lesson07.grid" />
        </block>
    </lesson07admin_adminhtml_lesson07_grid>
```

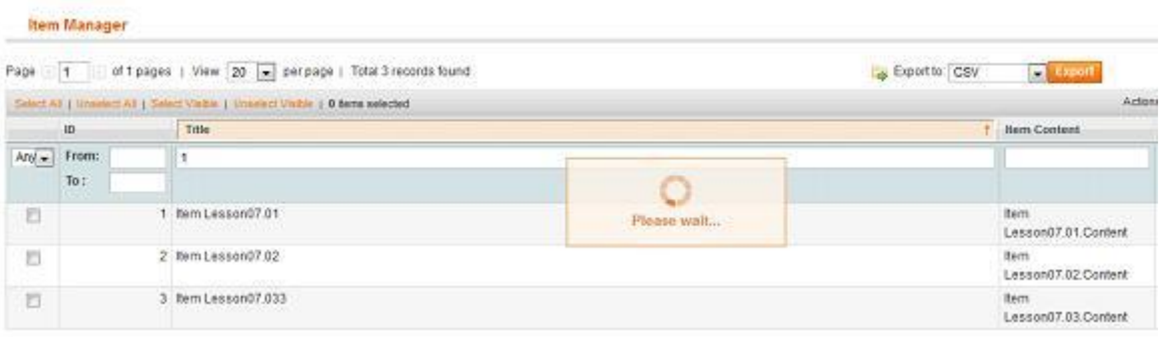
- *Editing Grid file to add the Ajax attributes*

File: app/code/local/Magestore/Lesson07/Block/Adminhtml/Lesson07/Grid.php

```
public function __construct()
    {
        parent::__construct();
        …..
        $this->setUseAjax(true);
    }
```

```
public function getGridUrl()
    {
        return $this->getUrl('*/*/grid', array('_current'=>true));
    }
```

Result:


Item Manager
Page 1 of 1 pages | View 20 per page | Total 3 records found
Export to: CSV
Export
Select All | Unselect All | Select Visible | Unselect Visible | 0 items selected
Actions
Please wait...

	ID	Title	Item Content
Any	From: To:	1	
☐	1	Item Lesson07.01	Item Lesson07.01.Content
☐	2	Item Lesson07.02	Item Lesson07.02.Content
☐	3	Item Lesson07.033	Item Lesson07.03.Content

III. SUMMARY

After learning this lesson, you are expected to have knowledge of:

- What grid in block adminhtml is and how to create a grid
- Structure and template of a simple grid
- How to declare block grid adminhtml in layout
- How to create a block grid with database collection and grid columns.

Back to top

Lesson 08: Advanced Grid

I. OVERVIEW

1. Time: 2 hours

2. Content

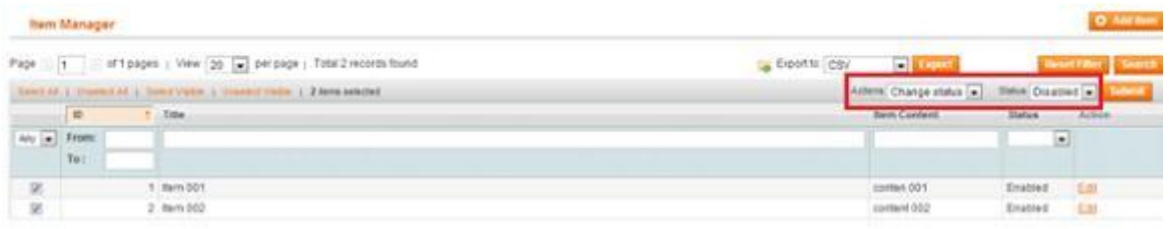
- Mass action in Magento grid
  - Displaying types in Magento grid
  - Supported-by-Magento types
  - Unsupported-by-Magento types
- Exporting data to CSV/XML in Magento grid

II. CONTENT

1. Mass action in Magento grid

1.1. Benefit of mass actions

This helps work with many rows at the same time without having to view each row in detail.


Item Manager
Add Item
Page 1 of 1 pages | View 20 per page | Total 2 records found
Export to: CSV
Export
Reset Filter
Search
Select All | Unselect All | Select Visible | Unselect Visible | 2 items selected
Actions: Change status
Status: Disabled
Submit
Any
From:
To:

	ID	Title	Item Content	Status	Action
☑	1	Item 001	content 001	Enabled	Edit
☑	2	Item 002	content 002	Enabled	Edit

1.2. Creating mass actions for Magento grid

- Step1: Write the function _prepareMassaction() in the file block grid

Example:

Code:*app\code\local\Magestore\Lesson08\Block\Adminhtml\Lesson08\Grid.php*

```
/**
     * prepare mass action for this grid
     *
     * @return Magestore_Lesson08_Block_Adminhtml_Lesson08_Grid
     */
    protected function _prepareMassaction()
    {
        $this->setMassactionIdField('lesson08_id');
        $this->getMassactionBlock()->setFormFieldName('lesson08');

        $this->getMassactionBlock()->addItem('delete', array(
            'label'        => Mage::helper('lesson08')->__('Delete'),
            'url'        => $this->getUrl('*/*/massDelete'),
            'confirm'    => Mage::helper('lesson08')->__('Are you sure?')
        ));

        $statuses = Mage::getSingleton('lesson08/status')-
>getOptionArray();

        array_unshift($statuses, array('label'=>'', 'value'=>''));
        $this->getMassactionBlock()->addItem('status', array(
            'label'=> Mage::helper('lesson08')->__('Change status'),
            'url'    => $this->getUrl('*/*/massStatus',
array('_current'=>true)),
            'additional' => array(
                'visibility' => array(
                    'name'    => 'status',
                    'type'    => 'select',
                    'class'    => 'required-entry',
                    'label'    => Mage::helper('lesson08')->__('Status'),
                    'values'=> $statuses
                ))
        ));
        return $this;
    }
```

- ✓ *setMassactionIdField:* set id field for mass action
- ✓ *setFormFieldName:* set name for the field of posting data

Example:

```
$this->getMassactionBlock()->setFormFieldName('lesson08');
```

The System will post an ID array with the name of 'lesson08'.

- ✓ *addItem:* add one mass action

```
    /**
     * Add new massaction item
     *
     * $item = array(
     *       'label'    => string,
     *       'complete' => string, // Only for ajax enabled grid (optional)
     *       'url'      => string,
     *       'confirm'  => string, // text of confirmation of this action
(optional)
     *       'additional' => string|array|Mage_Core_Block_Abstract //
(optional)
     * );
     *
     * @param string $itemId
     * @param array $item
     * @return Mage_Adminhtml_Block_Widget_Grid_Massaction_Abstract
     */
    public function addItem($itemId, array $item)
    {
        $this->_items[$itemId] =  $this->getLayout()-
>createBlock('adminhtml/widget_grid_massaction_item')
            ->setData($item)
            ->setMassaction($this)
            ->setId($itemId);

        if($this->_items[$itemId]->getAdditional()) {
            $this->_items[$itemId]->setAdditionalActionBlock($this-
>_items[$itemId]->getAdditional());
            $this->_items[$itemId]->unsAdditional();
        }

        return $this;
    }
```

In this example, we have just added two mass actions:

• *massDelete* to delete a list of rows

• *massStatus* to change status of many rows at the same time

<u>- Step 2</u>: Write action in the file controller to carry out this action

Example:

Code: *app\code\local\Magestore\Lesson08\controllers\Adminhtml\Lesson08Controller.php*

- ***Mass Delete***

```
public function massDeleteAction()
    {
        $lesson08Ids = $this->getRequest()->getParam('lesson08');
        if (!is_array($lesson08Ids)) {
            Mage::getSingleton('adminhtml/session')-
>addError(Mage::helper('adminhtml')->__('Please select item(s)'));
        } else {
            try {
                foreach ($lesson08Ids as $lesson08Id) {
                    $lesson08 = Mage::getModel('lesson08/lesson08')-
>load($lesson08Id);
                    $lesson08->delete();
                }
                Mage::getSingleton('adminhtml/session')->addSuccess(
                    Mage::helper('adminhtml')->__('Total of %d record(s)
were successfully deleted',
                    count($lesson08Ids))
                );
            } catch (Exception $e) {
                Mage::getSingleton('adminhtml/session')->addError($e-
>getMessage());
            }
        }
        $this->_redirect('*/*/index');
    }
```

- ***Mass change status***

```
public function massStatusAction()
    {
        $lesson08Ids = $this->getRequest()->getParam('lesson08');
        if (!is_array($lesson08Ids)) {
            Mage::getSingleton('adminhtml/session')->addError($this-
>__('Please select item(s)'));
        } else {
            try {
                foreach ($lesson08Ids as $lesson08Id) {
                    Mage::getSingleton('lesson08/lesson08')
                        ->load($lesson08Id)
                        ->setStatus($this->getRequest()-
>getParam('status'))
                        ->setIsMassupdate(true)
                        ->save();
                }
                $this->_getSession()->addSuccess(
                    $this->__('Total of %d record(s) were successfully
updated', count($lesson08Ids))
                );
            } catch (Exception $e) {
                $this->_getSession()->addError($e->getMessage());
            }
        }
        $this->_redirect('*/*/index');
    }
```

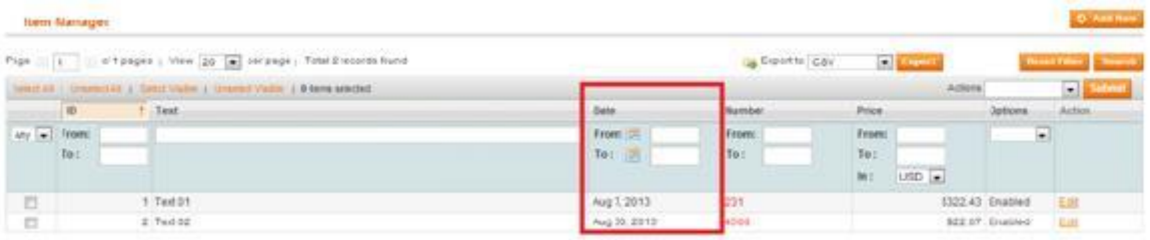
2. Displaying types in Magento grid

Example code: *app\code\local\Magestore\Lesson08\Block\Adminhtml\Lesson08\Grid.php*

By default, Magento supports many displaying types for data in grid such as number, text, date, currency, and option. These are the most popular ones:

2.1. Supported-by-Magento types

- Date



Renderer class: *adminhtml/widget_grid_column_renderer_date*

For example:

```
        $this->addColumn('date_field', array(
            'header'     => Mage::helper('lesson08')->__('Date'),
            'width'      => '150px',
            'index'      => 'date_field',
            'type'       => 'date',
            'format'     => Mage::app()->getLocale()-
>getDateFormat(Mage_Core_Model_Locale::FORMAT_TYPE_MEDIUM)
        ));
```

Some date format types:

- Mage_Core_Model_Locale::FORMAT_TYPE_MEDIUM

  Normal display

  Example, Aug 30, 2013

- Mage_Core_Model_Locale::FORMAT_TYPE_SHORT

  Short display

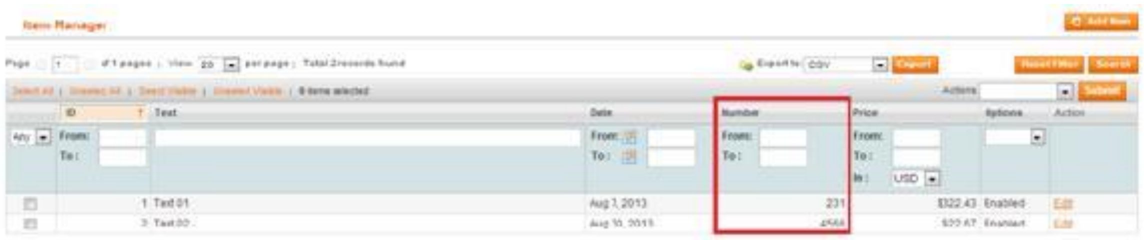
  Example, 8/30/13

- Mage_Core_Model_Locale::FORMAT_TYPE_LONG

  Full display

  Example, August 30, 2013

- *Number*



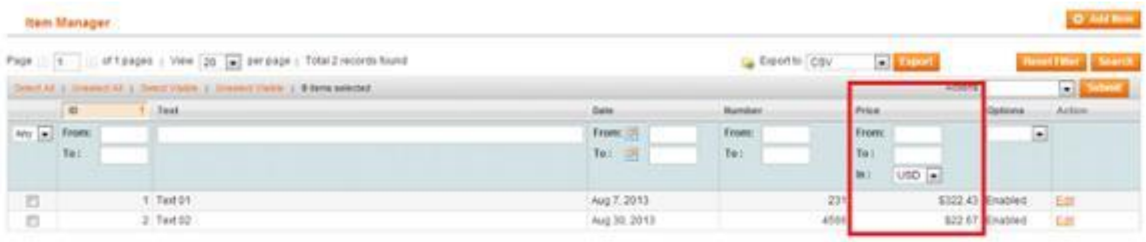
	ID	Text	Date	Number	Price	Options	Action
	1	Text 01	Aug 7, 2013	231	$322.43	Enabled	Edit
	2	Text 02	Aug 30, 2013	4566	$22.67	Enabled	Edit

Renderer class: *adminhtml/widget_grid_column_renderer_number*

For example:

```
$this->addColumn('number_field', array(
            'header'     => Mage::helper('lesson08')->__('Number'),
            'width'      => '150px',
            'index'      => 'number_field',
            'type'       => 'number'
        ));
```

- *Price*



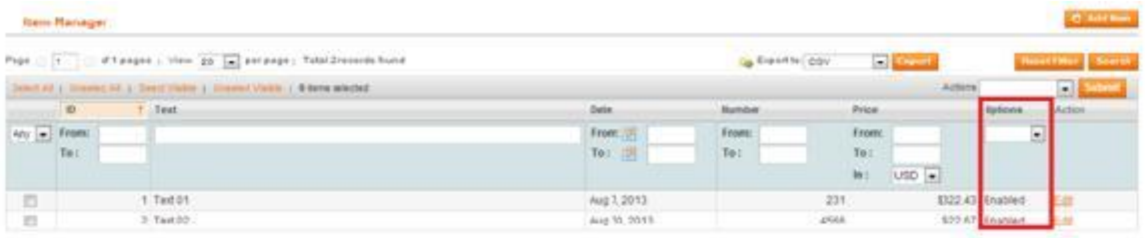
	ID	Text	Date	Number	Price	Options	Action
	1	Text 01	Aug 7, 2013	231	$322.43	Enabled	Edit
	2	Text 02	Aug 30, 2013	4566	$22.67	Enabled	Edit

Renderer class: *adminhtml/widget_grid_column_renderer_price*

For example:

```
        $this->addColumn('price_field', array(
            'header'    => Mage::helper('lesson08')->__('Price'),
            'width'     => '150px',
            'index'     => 'price_field',
            'type'      => 'price'
        ));
```

- Options



Page 1 of 1 pages | View 20 per page | Total 2 records found

	ID	Text	Date	Number	Price	Options	Action
	1	Text 01	Aug 7, 2013	231	$322.43	Enabled	Edit
	2	Text 02	Aug 10, 2013	4566	$22.47	Enabled	Edit

Renderer class: *adminhtml/widget_grid_column_renderer_options*

For example:

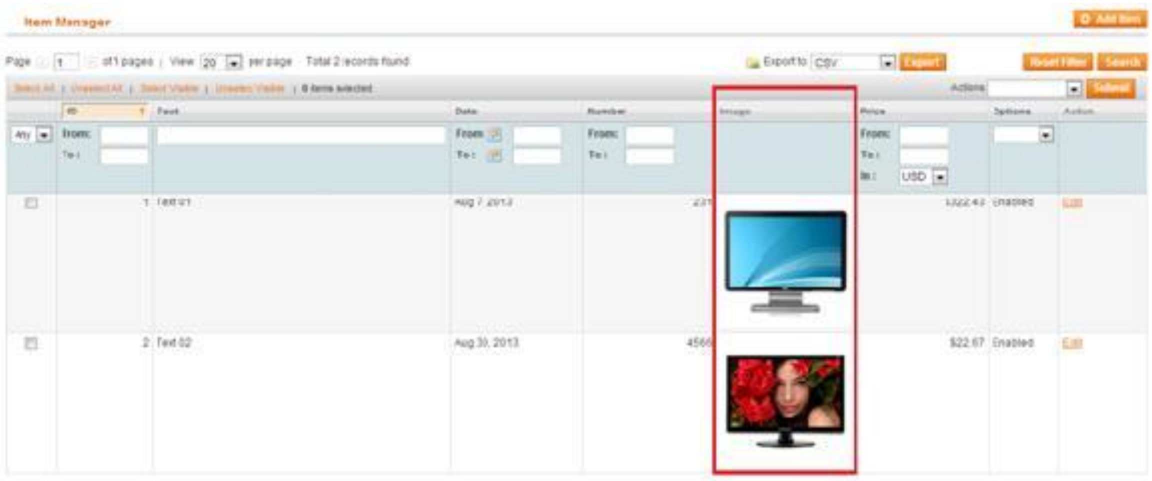
```
$this->addColumn('select_field', array(
            'header'    => Mage::helper('lesson08')->__('Options'),
            'align'     => 'left',
            'width'     => '80px',
            'index'     => 'status',
            'type'        => 'options',
            'options'     => array(
                1 => 'Enabled',
                2 => 'Disabled',
            ),
        ));
```

Besides, there are many other display types such as Country, Concat, Currency, Action, Checkbox, Datetime, Radio, Input, Select, Text and Store.

2.2. Unsupported-by-Magento types (Custom)

In reality, there are many types of display which have not been supported by Magento but still need to be shown such as images, email, and so on. In those cases, we will use renderer to add a new row whose display type can be adaptable (for example, sometimes it will be displayed as number; some other times, it will be shown as text).

There is a simple way to solve the above problem that is to create a class inheriting from the class **Mage_Adminhtml_Block_Widget_Grid_Column_Renderer_Abstract**, rewrite the render method for thfis new block and declare custom renderer in the method**_prepareColumns()** for the rendered fields.


Item Manager
Add Item
Page 1 of 1 pages | View 20 per page Total 2 records found
Export to CSV Export
Reset Filter Search
Select All | Unselect All | Select Visible | Unselect Visible | 0 items selected
Actions Submit

	ID	Text	Date	Number	Image	Price	Options	Action
	1	Text 01	Aug 7, 2013	23		$322.43	Enabled	Edit
	2	Text 02	Aug 30, 2013	4566		$22.67	Enabled	Edit

In this example, we will create a field to display image.

Example:

Code: app\code\local\Magestore\Lesson08\Block\Adminhtml\Lesson08\Grid.php

__Step 1__: Declare renderer for the field in the method _prepareColumns()

```
        $this->addColumn('filename', array(
            'header'     => Mage::helper('lesson08')->__('Image'),
            'width'      => '150px',
            'index'      => filename',
            'renderer'   => 'lesson08/adminhtml_lesson08_renderer_image'
        ));
```

__Step 2__**:** Create class renderer inheriting from the class

Mage_Adminhtml_Block_Widget_Grid_Column_Renderer_Abstract

__Step 3__: Write render function for the class that has just been created.

For instance:

Code:app\code\local\Magestore\Lesson08\Block\Adminhtml\Lesson08\Renderer\Image.php

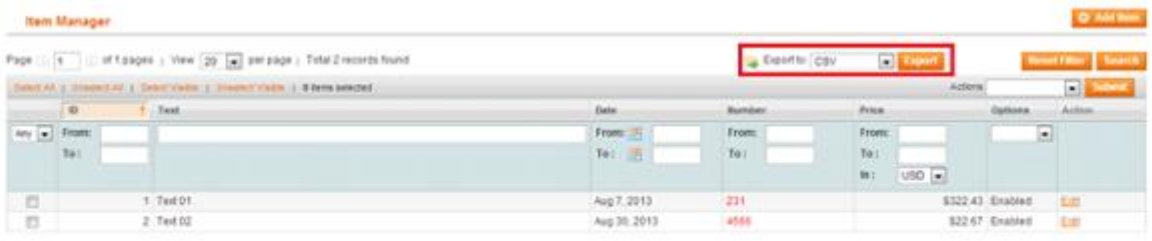
```
class Magestore_Lesson08_Block_Adminhtml_Lesson08_Renderer_Image extends
Mage_Adminhtml_Block_Widget_Grid_Column_Renderer_Abstract {

    public function render(Varien_Object $row) {
        $value = $row->getData($this->getColumn()->getIndex());
        return '<img width="150" src="' .
Mage::getBaseUrl('media').'lesson08/'.$value . '" />';
    }


}
```

3. Exporting data from Magento grid to CSV/ XML files

Example:

Code: ***app\code\local\Magestore\Lesson08\Block\Adminhtml\Lesson08\Grid.php***

This function helps export data from Magento grid to CSV/XML files in order to print or import them into other accounting system.



<u>**Step 1**</u>***:*** **Add** the box "Select" and the button "Export" to the grid through the function**_prepareColumns()**

```
$this->addExportType('*/*/exportCsv', Mage::helper('lesson08')->__('CSV'));
        $this->addExportType('*/*/exportXml', Mage::helper('lesson08')-
>__('XML'));
```

In the function addExportType:

- First parameter: action to process data exportation

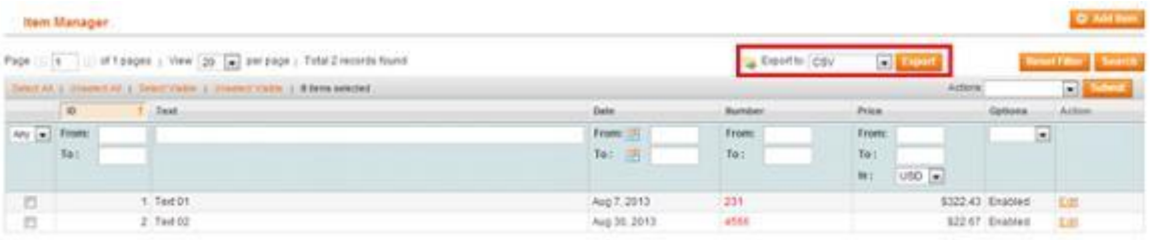
For example, `'*/*/exportCsv'`

In this example, the action to process exporting data to CSV is **exportCSV** in the controller file ***app\code\local\Magestore\Lesson08\controllers\Adminhtml\Lesson08Controller.php***

- Second parameter: label of export type

For example: `Mage::helper('lesson08')->__('CSV')`

This label will be displayed in the select box of export


Item Manager
Add Item
Page 1 of 1 pages | View 20 per page | Total 2 records found
Export to: CSV
Export
Reset Filter
Search
Select All | Unselect All | Select Visible | Unselect Visible | 0 items selected
Actions
Submit
Any
From:
To:
In: USD

	ID	Text	Date	Number	Price	Options	Action
☐	1	Text 01	Aug 7, 2013	231	$322.43	Enabled	Edit
☐	2	Text 02	Aug 30, 2013	4566	$22.67	Enabled	Edit

<u>Step 2</u>*:* Write action for each type in the controller file

Example code:

app\code\local\Magestore\Lesson08\controllers\Adminhtml\Lesson08Controller.php

```
public function exportCsvAction()
    {
        $fileName   = 'lesson08.csv';
        $content    = $this->getLayout()
                                     -
>createBlock('lesson08/adminhtml_lesson08_grid')
                                     ->getCsv();
        $this->_prepareDownloadResponse($fileName, $content);
    }
```

In this example, after exporting to CSV, we get a file as below:

```
"ID","Text","Date","Number","Price","Options"
"1","Text 01","Aug 7, 2013","231","$322.43","Enabled"
"2","Text 02","Aug 30, 2013","4566","$22.67","Enabled"
```

III. SUMMARY

Having finished this lesson, you should grasp these points:

- Mass actions in Magento grid
- Display types in grid
- Exporting data from grid to CSV/XML files.

<u>Back to top</u>

Lesson 09: Form in Magento Backend

I. INTRODUCTION

1. Time: 2-4 hours

2. Content

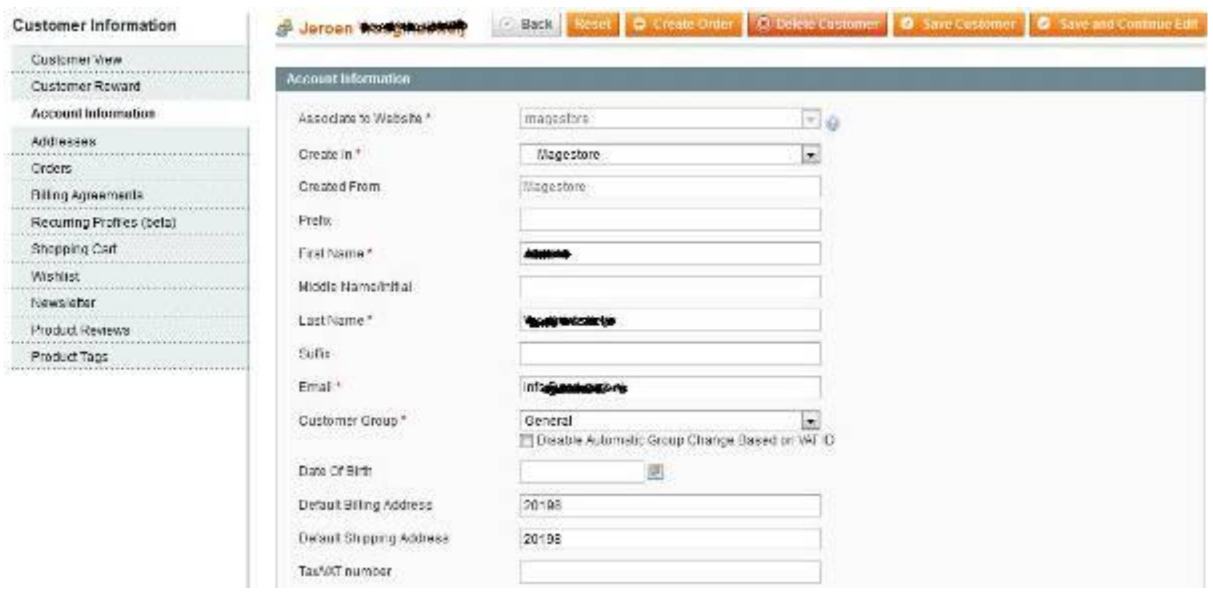
- Magento form and its elements in backend
- Creating a Magento form in backend
- Actions in a Magento form (save, delete, save and continue edit)

II. CONTENT

1. Magento form and its elements in backend

Form allows users to enter and save data in the database. A Magento form includes five elements: Form container, Form tag, Form tabs, Form action button, and Form fields.

Below is how the Customer Account Information form looks:



Associate to Website *	magestore
Create In *	Magestore
Created From	Magestore
Prefix	
First Name *	[redacted]
Middle Name/Initial	
Last Name *	[redacted]
Suffix	
Email *	inf[redacted]
Customer Group *	General
	☐ Disable Automatic Group Change Based on VAT ID
Date Of Birth	
Default Billing Address	20198
Default Shipping Address	20198
Tax/VAT number	

2. Creating a Magento form in backend

2.1 Creating action to display the Magento form in backend

- Add new item case

In the case we need to add new item, we should write function newAction() in the controller file:

app/code/local/Magestore/Lesson09/controllers/Adminhtml/Lesson09Controller.php

```
public function newAction(){
    $this->loadLayout();
     $this->_setActiveMenu('lesson09/items');
     $this->_addBreadcrumb(Mage::helper('adminhtml')->__('Item Manager'),
Mage::helper('adminhtml')->__('Item Manager'));
     $this->_addBreadcrumb(Mage::helper('adminhtml')->__('Item News'),
Mage::helper('adminhtml')->__('Item News'));

    $this->_addContent($this->getLayout()->createBlock('
lession09/adminhtml_lession09_edit'))
           ->_addLeft($this->getLayout()
           ->createBlock('lession09/adminhtml_lession09_edit_tabs'));
    $this->renderLayout();
}
```

On the other hand, we can use a shot function. However, it can be used only when we had editAction() in this controller. Look at the edit item case for better understanding.

```
    public function newAction() {
        $this->_forward('edit');
    }
```

- Edit item case

We need to add some code to load data from the database of the item edited:

```
public function editAction() {
    $id     = $this->getRequest()->getParam('id');
    $model  = Mage::getModel('lesson09/lesson09')->load($id);

    if ($model->getId() || $id == 0) {
        $data = Mage::getSingleton('adminhtml/session')->getFormData(true);
        if (!empty($data)) {
            $model->setData($data);
        }

        Mage::register('lesson09_data', $model);
        $this->loadLayout();
        $this->_setActiveMenu('lesson09/items');


        $this->_addBreadcrumb(Mage::helper('adminhtml')->__('Item
Manager'), Mage::helper('adminhtml')->__('Item Manager'));
        $this->_addBreadcrumb(Mage::helper('adminhtml')->__('Item News'),
Mage::helper('adminhtml')->__('Item News'));


        $this->_addContent($this->getLayout()->createBlock('
lession09/adminhtml_lession09_edit'))
          ->_addLeft($this->getLayout()
          ->createBlock('lession09/adminhtml_lession09_edit_tabs'));
        $this->renderLayout();
    } else {
        Mage::getSingleton('adminhtml/session')-
>addError(Mage::helper('lesson09')->__('Item does not exist'));
        $this->_redirect('*/*/');
    }
}
```

2.2 Creating blocks in Magento backend

- Form container

The form container is, as the name suggests, the main div tag where all the form elements/html are placed. To create a form container, you need to create a PHP file in your Block/Adminhtml folder which declares a class extended from Mage_Adminhtml_Block_Widget_Form_Container class. Name container file as Form.php app/code/local/Magestore/Lesson09/Block/Adminhtml/Lesson09/Edit.php

```

class Magestore_Lesson09_Block_Adminhtml_Lesson09_Edit extends
Mage_Adminhtml_Block_Widget_Form_Container
{
    public function __construct()
    {
        parent::__construct();


        $this->_objectId = 'id';
        $this->_blockGroup = 'lesson09';
        $this->_controller = 'adminhtml_lesson09';


        $this->_updateButton('save', 'label', Mage::helper('lesson09')-
>__('Save'));
        $this->_updateButton('delete', 'label', Mage::helper('lesson09')-
>__('Delete'));

        $this->_addButton('saveandcontinue', array(
            'label'     => Mage::helper('adminhtml')->__('Save And Continue
Edit'),
            'onclick'   => 'saveAndContinueEdit()',
            'class'     => 'save',
        ), -100);
    }

    public function getHeaderText()
    {
        return Mage::helper('lesson09')->__('My Form Container');
    }
}
```

Note:

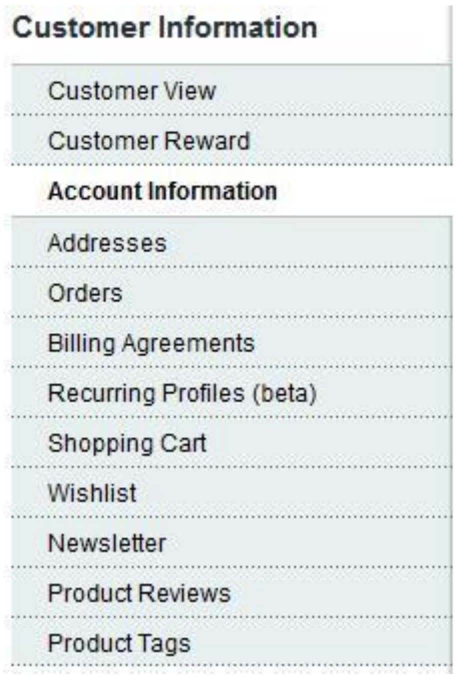
- getHeaderText(): This function returns the text to display as the form header. You can see the form header in the screenshot attached Admin Form Container.
- $this->_objectId: This variable is used in the form URL's. It has the forms entity primary key; for example, the delete button URL would be module/controller/action/$this->_objectid/3.
- $this->_blockGroup = 'lesson09′; and $this->_controller = 'adminhtml_lesson09': They are very important, these variables are used to find FORM tabs file. For example, the path of the form tabs should be {$this->_blockGroup . '/' . $this->_controller . '_' . $this->_mode . '_form'}. The value of $this->_mode by default is 'edit'. So the path of the PHP file which contains the form tag in our case would be 'lesson09/adminhtml_lesson09_edit_form'.
- $this->_updateButton() and $this->_addButton() are used to update/add buttons to the form container. These buttons are visible in the screenshot Admin Form Container

- Form tag

This PHP file contains the actual <form> tag. It is extended from Mage_Adminhtml_Block_Widget_Form. Full path of this file is app/code/local/Magestore/Lesson09/Block/Adminhtml/Lesson09/Edit/Form.php

```
class Excellence_Lesson09_Block_Adminhtml_Lesson09_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',
                                      'enctype' => 'multipart/form-data'
                                   )
      );

      $form->setUseContainer(true);
      $this->setForm($form);
      return parent::_prepareForm();
  }
}
```

- Form tabs



The class of this PHP would be Magestore_Lesson09_Block_Adminhtml_Lesson09_Edit_Tabs, this is the block class we had used in our controller file. The full patch is

App/code/local/Magestore/Lesson09/Block/Adminhtml/Lesson09/Edit/Tabs.php

```
class Magestore_Lesson09_Block_Adminhtml_Lesson09_Edit_Tabs extends
Mage_Adminhtml_Block_Widget_Tabs
{

  public function __construct()
  {
      parent::__construct();
      $this->setId('form_tabs');
      $this->setDestElementId('edit_form');
      $this->setTitle(Mage::helper('lesson09')->__('Lesson09
Information'));
  }

  protected function _beforeToHtml()
  {
      $this->addTab('form_section', array(
          'label'     => Mage::helper('lesson09')->__('Item Information'),
          'title'     => Mage::helper('lesson09')->__('Item Information'),
          'content'   => $this->getLayout()-
>createBlock('lesson09/adminhtml_lesson09_edit_tab_form')->toHtml(),
      ));

      return parent::_beforeToHtml();
  }
}
```

- Form fields

This block supports adding fields to the form. It is extended from Mage_Adminhtml_Block_Widget_Form. The full patch is

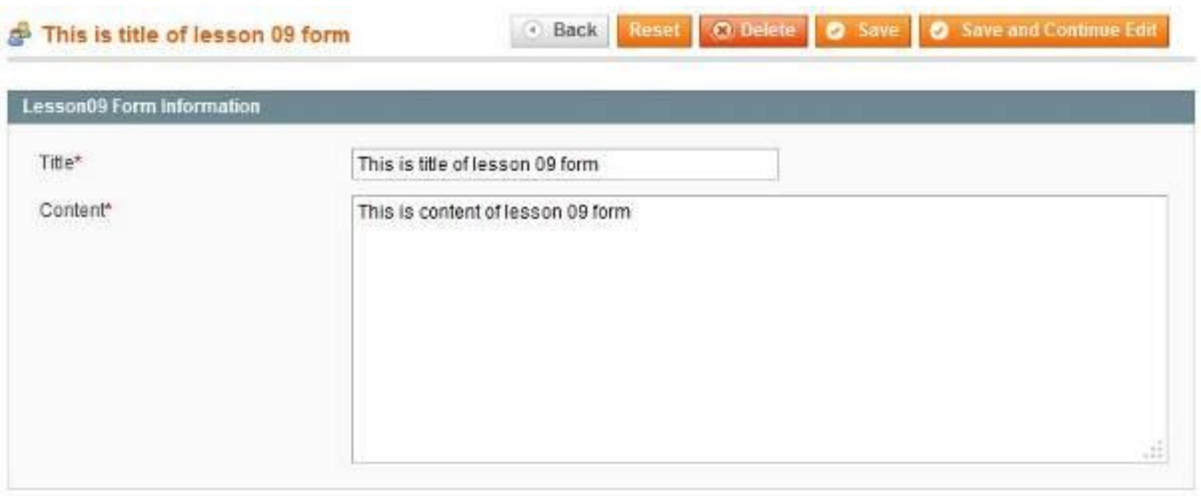
app/code/local/Magestore/Lesson09/Block/Adminhtml/Lesson09/Edit/Tab/Form.php

```
class Magestore_Lesson09_Block_Adminhtml_Lesson09_Edit_Tab_Form extends
Mage_Adminhtml_Block_Widget_Form
{
    protected function _prepareForm()
    {
        $form = new Varien_Data_Form();
        $this->setForm($form);
        $fieldset = $form-
>addFieldset('lession09_form',array('legend'=>Mage::helper('lession09')-
>__('Item information')));


        $fieldset->addField('title', 'text', array(
          'label'     => Mage::helper('lession09')->__('Title'),
          'class'     => 'required-entry',
          'required'  => true,
          'name'      => 'title',
        ));

        $fieldset->addField('content', 'editor', array(
          'name'      => 'content',
          'label'     => Mage::helper('dochelp')->__('Content'),
          'title'     => Mage::helper('dochelp')->__('Content'),
          'style'     => 'width:700px; height:500px;',
          'wysiwyg'   => false,
          'required'  => true,
        ));

        if ( Mage::getSingleton('adminhtml/session')->getlesson09Data() )
        {
          $form->setValues(Mage::getSingleton('adminhtml/session')-
>getlesson09Data());
          Mage::getSingleton('adminhtml/session')->setlesson09Data(null);
        } elseif ( Mage::registry('lesson09_data') ) {
          $form->setValues(Mage::registry('lesson09_data')->getData());
        }



        return parent::_prepareForm();
    }
}
```

We have added two fields to this form: title with text type (input text element) and content with editor type (text area element).

<u>Note</u>: Title and content are name of those attribute in database.

Below is the screenshot of form we have created:



Lesson09 Form Information

Title* This is title of lesson 09 form

Content* This is content of lesson 09 form

3. Actions in a Magento form

3.1. Save Action

The **Save** action is written in the controller file:

app/code/local/Magestore/Lesson09/controllers/Adminhtml/Lesson09Controller.php

```
public function saveAction() {
    if ($data = $this->getRequest()->getPost()) {

        $model = Mage::getModel('lesson09/lesson09');
        $model->setData($data)
            ->setId($this->getRequest()->getParam('id'));

        try {
            if ($model->getCreatedTime == NULL || $model->getUpdateTime()
== NULL) {
                $model->setCreatedTime(now())
                    ->setUpdateTime(now());
            } else {
                $model->setUpdateTime(now());
            }

            $model->save();
            Mage::getSingleton('adminhtml/session')-
>addSuccess(Mage::helper('lesson09')->__('Item was successfully saved'));
            Mage::getSingleton('adminhtml/session')->setFormData(false);

            if ($this->getRequest()->getParam('back')) {
                $this->_redirect('*/*/edit', array('id' => $model-
>getId()));
                return;
            }
            $this->_redirect('*/*/');
            return;
        } catch (Exception $e) {
            Mage::getSingleton('adminhtml/session')->addError($e-
>getMessage());
            Mage::getSingleton('adminhtml/session')->setFormData($data);
            $this->_redirect('*/*/edit', array('id' => $this->getRequest()-
>getParam('id')));
            return;
        }
    }
    Mage::getSingleton('adminhtml/session')-
>addError(Mage::helper('lesson09')->__('Unable to find item to save'));
    $this->_redirect('*/*/');
}
```

3.2. Delete Action

The Delete action is also written in the controller file:

app/code/local/Magestore/Lesson09/controllers/Adminhtml/Lesson09Controller.php

```
public function deleteAction() {
    if( $this->getRequest()->getParam('id') > 0 ) {
        try {
            $model = Mage::getModel('lesson09/lesson09');

            $model->setId($this->getRequest()->getParam('id'))
                ->delete();

            Mage::getSingleton('adminhtml/session')-
>addSuccess(Mage::helper('adminhtml')->__('Item was successfully
deleted'));
            $this->_redirect('*/*/');
        } catch (Exception $e) {
            Mage::getSingleton('adminhtml/session')->addError($e-
>getMessage());
            $this->_redirect('*/*/edit', array('id' => $this->getRequest()-
>getParam('id')));
        }
    }
    $this->_redirect('*/*/');
}
```

3.3. Save and Continue Edit Action

This action includes two sub-actions: save and edit. It is not written in the controller and is just embedded in the Save Action with the code below:

```
if ($this->getRequest()->getParam('back')) {
        $this->_redirect('*/*/edit', array('id' => $model->getId()));
        return;
}
```

The “Save and Continue Edit” button is embedded in the Container Form with the code:

```
$this->_addButton('saveandcontinue', array(
            'label'     => Mage::helper('adminhtml')->__('Save And Continue
Edit'),
            'onclick'   => 'saveAndContinueEdit()',
            'class'     => 'save',
        ), -100);
    }
```

This command has added the param “**back**” to save URL. So after saving the data, we will be redirected to the edit action.

III. SUMMARY

Finishing this lesson, you can now create a simple form in your Magento website's backend. Lesson 10 will go further with specific input types created in form such as option, datetime, file, image and checkbox.

Back to top

Lesson 10: Advanced Form in Backend

I. OVERVIEW

In the previous lesson, we looked at form and fundamental elements of a form in Magento backend. This lesson will continue this topic but at a higher level.

1. Time: 3 hours

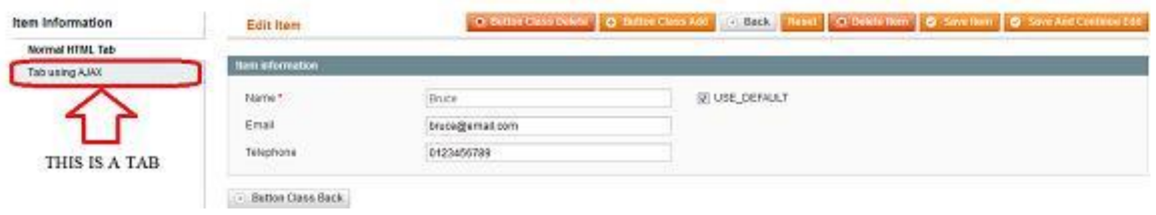
2. Content:

- Normal HTML tabs and AJAX-based tabs
- Renderer field in form
- Custom purpose button

II. CONTENT

1. Adding Magento tabs

A tab in Magento:



Name * Bruce ☑ USE_DEFAULT
Email bruce@email.com
Telephone 0123456789

1.1. Adding normal HTML tabs

To add new HTML tabs, you should notice the file Tabs.php. You can easily find this file through the link

app/code/local/{your_namespace}/{your_module}/block/adminhtml/{section}/edit/

After opening the file Tabs.php, you can see:

```
class Magestore_Lesson10_Block_Adminhtml_Lesson10_Edit_Tabs extends
Mage_Adminhtml_Block_Widget_Tabs
{
    public function __construct()
    {
        parent::__construct();
        $this->setId('lesson10_tabs');
        $this->setDestElementId('edit_form');
        $this->setTitle(Mage::helper('lesson10')->__('Item
Information'));
    }
    /**
     * prepare before render block to html
     *
     * @return Magestore_Lesson10_Block_Adminhtml_Lesson10_Edit_Tabs
     */
    protected function _beforeToHtml()
    {
      $this->addTab('form_section', array(
        'label' => Mage::helper('lesson10')->__('Normal HTML Tab'),
        'title' => Mage::helper('lesson10')->__('Normal HTML Tab'),
        'content' => $this->getLayout()
           ->createBlock('lesson10/adminhtml_lesson10_edit_tab_form')
           ->toHtml(),
        ));

    }
}
```

You should note following points:

- function addTab() that helps you add normal HTML tabs includes two parameters:
    - tabId: Unique Id for tab. The ID of tab here is "form_section"
    - tab: Array containing the declaration about tab's attributes. Also, you can also use an object or a string.

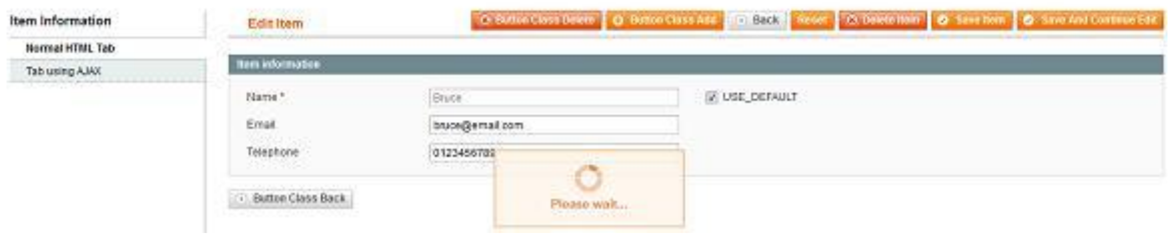
You are recommended to go to the parent class Mage_Adminhtml_Block_Widget_Tabs to know more about this method.

- The attributes declared in these arrays
    - 'Label': Display name of tab
    - 'Title': Title of tab
    - 'Content': It defines the content in a tab. Tab's content is a HTML. You can totally change the link in the function createBlock() to change the tab's content.

From now on, you can add as many normal HTML tabs as you want in a similar way.

1.2. Adding AJAX based tabs

The difference between an Ajax-based tab and a normal HTML tab is that Ajax-based tab uses AJAX to load content of tab. The tab's content is loaded only when you click on the tab. After you click on an AJAX tab, the waiting icon will appear while AJAX is loading the content. In reality, Ajax-based tab is often used when you want to add a grid to tab. For example:



Name * Bruce ☑ USE_DEFAULT
Email bruce@email.com
Telephone 0123456789

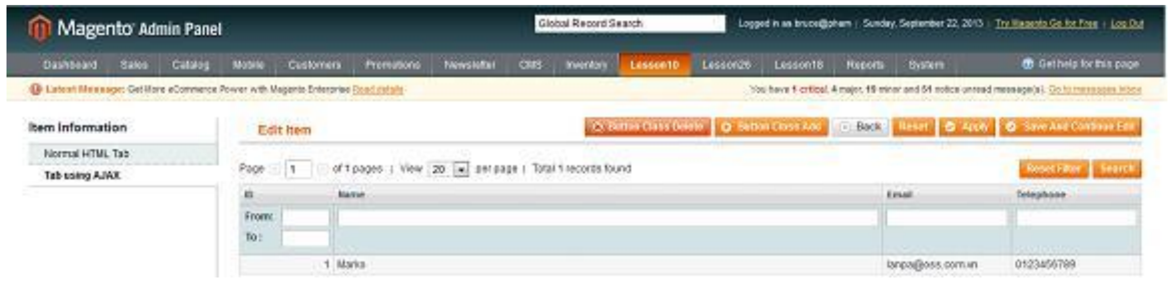
Continue with the file Tabs.php, the following code will help you add an AJAX tab to form:

```
$this->addTab('ajax_base_tab', array(
      'label'  => Mage::helper('lesson10')->__('Tab using AJAX'),
      'url'    => $this->getUrl('*/*/ajaxTab', array('_current' =>
true)),
      'class'  => 'ajax',
));
return parent::_beforeToHtml();
```

We have named the class "ajax" and added an URL to AJAX to load. This URL will call the function ajaxTabAction() in the admin controller. This function will give out the tab's content.

```
public function ajaxTabAction(){
    $this->loadLayout();
    $this->getResponse()->setBody(
    $this->getLayout()
        ->createBlock('lesson10/adminhtml_lesson10_edit_tab_ajaxtab');
        ->toHtml()
    );
}
```

Basically, tab content is a HTML code. Based on the objective of using tab, we should flexibly use the normal or AJAX tab.



Latest Message: Get More eCommerce Power with Magento Enterprise Read details
You have 1 critical, 4 major, 15 minor and 54 notice unread message(s). Go to messages inbox

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

ID	Name	Email	Telephone
From: To:			
1	Marks	lanpa@oss.com.vn	0123456789

2. Renderer field in form

2.1. Magento form fields

Magento has many different types of field available by default. So see the syntax of using each of these fields. All the different types of Magento form fields available are located in the folder lib\Varien\Data\Form\Element.

- Text

This is how to add a text field to an admin form with a list of options.

```
$fieldset->addField(name, 'text', array(
          'label'     => Mage::helper('form')->__('Name'),
          'class'     => 'required-entry',
          'required'  => true,
          'name'      => 'name',
          'onclick' => "alert('on click');",
          'onchange' => "alert('on change');",
          'style'   => "border:10px",
          'disabled' => false,
          'readonly' => true,
          'after_element_html' => '<b>Textfield</b>',
          ));
```

Properties in the above array:

– Label: The label of the field

– Class: Class of the element that was defined in CSS

– Required: Whether the field is required or not.

– Onclick: Response when you click on the element

– Onchange: Response when you change the value of the element

– Readonly: Whether the field can be edited or not

– Disabled: Whether the field is disabled or not

– After_element_html: What is right after element? It may be some texts or another element.

- Text area

```
$fieldset->addField('textarea', 'textarea', array(
            'label'     => Mage::helper('form')->__('TextArea'),
            'class'     => 'required-entry',
            'required'  => true,
            'name'      => 'title',
            'onclick' => "",
            'onchange' => "",
            'disabled' => false,
            'readonly' => false,
            'after_element_html' => '<b>Text area</b>',
        ));
```

- Note

```
    $fieldset->addField('note', 'note', array(
          'text'     => Mage::helper('form')->__('Text Text'),
    ));
```

- Date

```
    $fieldset->addField('date', 'date', array(
        'label'      => Mage::helper('form')->__('Date'),
        'after_element_html' => '<small>Date</small>',
        'image' => $this->getSkinUrl('images/grid-cal.gif'),
        'format' => Mage::app()->getLocale()-
    >getDateFormat(Mage_Core_Model_Locale::FORMAT_TYPE_MEDIUM)
            ));
```

Some properties you should know:

– Image: The image that is shown besides the field

– Format: The date format type. You can change it to FORMAT_TYPE_FULL, FORMAT_TYPE_SHORT, FORMAT_TYPE_LONG.

- Time

```
    $fieldset->addField('time', 'time', array(
          'label'     => Mage::helper('form')->__('Time'),
          'class'     => 'required-entry',
          'required'  => true,
          'name'      => 'time',
          'onclick' => "",
          'onchange' => "",
          'disabled' => false,
          'readonly' => false,
          'after_element_html' => '<b>Time</b>',
          'tabindex' => 1
    ));
```

- Drop down

```
    $fieldset->addField('select', 'select', array(
          'label'     => Mage::helper('form')->__('Select'),
          'class'     => 'required-entry',
          'required'  => true,
          'name'      => 'title',
          'onclick' => "",
          'onchange' => "",
          'values' => array('-1'=>'Please Select..','1' => 'Option1','2'
    => 'Option2', '3' => 'Option3'),
          'disabled' => false,
          'readonly' => false,
          'after_element_html' => '<b>Drop down</b>',
          'tabindex' => 1
            ));
```

Values are an array that contains all the options of the element. Those elements that have options must have this property

Another kind of drop-down:

```
    $fieldset->addField('select2', 'select', array(
          'label'     => Mage::helper('form')->__('Select
    Type2'),
          'class'     => 'required-entry',
          'required'  => true,
          'name'      => 'title',
          'onclick' => "",
          'onchange' => "",
          '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'),
               ),
          'disabled' => false,
          'readonly' => false,
          'after_element_html' => '<b>Drop down</b>',
          'tabindex' => 1
            ));
```

- Multiselect

```
     $fieldset->addField('multiselect', 'multiselect', array(
           'label'     => Mage::helper('form')->__('Select Type'),
           'class'     => 'required-entry',
           'required'  => true,
           'name'      => 'title',
           'onclick' => "return false;",
           'onchange' => "return false;",
           'value'  => '4',
           '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'
                   ),
           ),
           'disabled' => false,
           'readonly' => false,
           'after_element_html' => '<b>Multiselect</b>',
           'tabindex' => 1
         ));
```

- Checkbox

```
     $fieldset->addField('checkboxes', 'checkboxes', array(
          'label'     => Mage::helper('form')->__('Checkboxs'),
          'name'      => 'Checkbox',
          'values' => array(
         array('value'=>'1','label'=>'Checkbox1'),

         array('value'=>'2','label'=>'Checkbox2'),
        array('value'=>'3','label'=>'Checkbox3'),
          ),
          'onclick' => "",
          'onchange' => "",
          'value'  => '1',
          'disabled' => false,
          'after_element_html' => '<small>Comments</small>',
          'tabindex' => 1
             ));
```

- Radio button

```
     $fieldset->addField('radio2', 'radios', array(
           'label'     => Mage::helper('form')->__('Radios'),
           'name'      => 'title',
           'onclick' => "",
           'onchange' => "",
           'value'  => '2',
           'values' => array(

        array('value'=>'1','label'=>'Radio1'),

        array('value'=>'2','label'=>'Radio2'),

        array('value'=>'3','label'=>'Radio3')),
           'disabled' => false,
           'readonly' => false,
           'after_element_html' => '<b>Radios button</b>',
           'tabindex' => 1
       ));
```

- Password

```
     $fieldset->addField('password', 'password', array(
           'label'     => Mage::helper('form')->__('Password'),
           'class'     => 'required-entry',
           'required'  => true,
           'name'      => 'title',
           'onclick' => "",
           'onchange' => "",
           'style'   => "",
           'disabled' => false,
           'readonly' => false,
           'after_element_html' => '<b>Password</b>',
           ));
```

Another kind of password field:

```
     $fieldset->addField('obscure', 'obscure', array(
           'label'     => Mage::helper('form')->__('Obscure'),
           'class'     => 'required-entry',
           'required'  => true,
           'name'      => 'obscure',
           'onclick' => "",
           'onchange' => "",
           'style'   => "",
           'value'  => '123456789',
           'after_element_html' => '<b>Obscure</b>',
           ));
```

- Link

```
    $fieldset->addField('link', 'link', array(
          'label'     => Mage::helper('form')->__('Link'),
          'style'   => "",
          'href' => 'www.blog.magestore.com',
          'value'  => 'Magestore Blog',
          'after_element_html' => '<b>Link</b>'
            ));
```

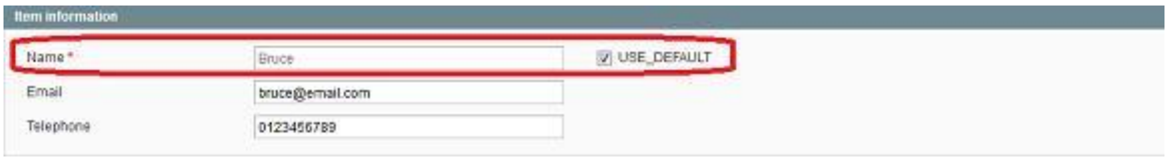
href: the reference link

- File upload

```
    $fieldset->addField('file', 'file', array(
          'label'     => Mage::helper('form')->__('Upload'),
          'value'  => 'Upload',
          'disabled' => false,
          'readonly' => true,
          'after_element_html' => '<b>File upload</b>',
          'tabindex' => 1
            ));
```

2.2. Custom form fields

In Magento form, if you are not satisfied with available fields of Magento, you can modify them as you wish by using renderer for form field:



Item information

Name *	Bruce	☑ USE_DEFAULT
Email	bruce@email.com	
Telephone	0123456789	

In Magento core system, there is no form field with a checkbox to enable or disable as seen from the above picture. It is simple to create that field. Follow the link *app/code/local/{your_namespace}/{your_module}/block/adminhtml/{section}/edit/tab* to find the file Form.php

For example:

```
$fieldset->addField('name', 'text', array(
            'label' => Mage::helper('lesson10')->__('Name'),
            'class' => 'required-entry',
            'required' => true,
            'name' => 'name',
            'onclick' => "alert('on click');",
            'onchange' => "alert('on change');",
            'disabled' => false,
            'readonly' => true,
            'after_element_html' => '<br/><b>Textfield</b>',
            'tabindex' => 1
        ));
     $form->getElement('name')->setRenderer(
          Mage::app()->getLayout()->createBlock(
          'lesson10/adminhtml_lesson10_edit_tab_renderer_name'));
```

To create a renderer, you need to take two steps:

Step 1: Get the field to render and set renderer for it. Here I choose field "name":

```
$form->getElement('name')->setRenderer(Mage::app()->getLayout()
->createBlock(lesson10/adminhtml_lesson10_edit_tab_renderer_name);
```

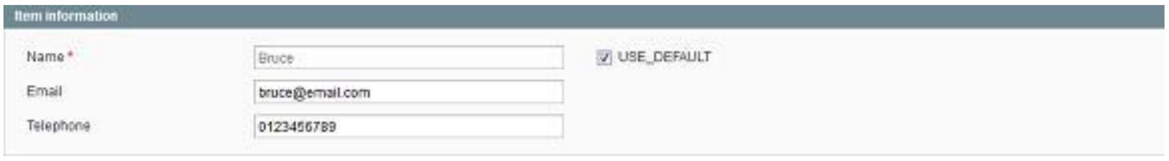
Step 2: **Create a renderer class in the file corresponding with the link in function** createBlock().

The link of this file is

app/code/local/magestore/lesson10/block/adminhtml/lesson10/edit/tab/renderer/name.php. Function render will be declared in this class

```
class
Magestore_Lesson10_Block_Adminhtml_Lesson10_Edit_Tab_Renderer_Name
extends Mage_Adminhtml_Block_Widget implements
Varien_Data_Form_Element_Renderer_Interface {

public function render(Varien_Data_Form_Element_Abstract $element) {
     $element->setDisabled(true);
     $disabled = true;
     $title = 'use_default';
     $html  = '<td class="label">'.$element->getLabelHtml().'</td>';
     $html.= '<td class = "value">'
.$element->getElementHtml().'</td>';
     $html.= '</td><td class="use-default">
         <input id="package_name_default" name="package_name_default"
type="checkbox" value="1" class="checkbox config-inherit" '
             .($disabled ? 'checked="checked"' : '').'
onclick="toggleValueElements(this,
Element.previous(this.parentNode))" />
         <label for="package_name_default" class="inherit"
title="'.$title.'">'.$title.'</label>
       </td>';
     return $html;
}
```

Now there appears a checkbox next to the "name" field to enable and disable it.

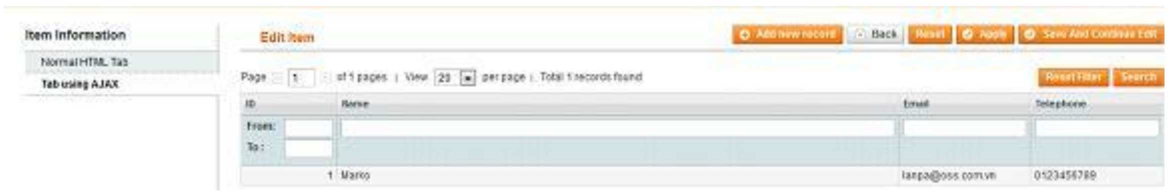

Item information

Name *	Bruce	☑ USE_DEFAULT
Email	bruce@email.com	
Telephone	0123456789	

3. Custom buttons

3.1. Adding custom buttons

In the Magento form container, you can add one or more buttons for different purposes. Below is the example image in form with the newly-added button "Add new record":



ID	Name	Email	Telephone
1	Marko	lanpa@oss.com.vn	0123456789

And this is the instruction:

```
      $this->_addButton('btn_add', array(
          'label'      => Mage::helper('adminhtml')->__(Add new
record'),
          'onclick'    => 'setLocation(\'' . $this->getUrl('*/*/new')
.'\')',
          'class'      => 'add',
        ),-1,3);
```

To make it clearer for you, the above code is added to the Edit.php file. You can find this file following the link:

*app/code/local/{your_namespace}/{your_module}/block/adminhtml/{section}***.**

I will explain a bit about parameters of the function _addButton():

```
protected function _addButton($id, $data, $level = 0, $sortOrder =
100, $area = 'header'){...}
```

- *$id*: button id is the unique ID for each button

- *$data*: button parameter array, you can set attributes label, on-click and class for each button. Class can receive such values as delete, save, back, add, etc. Button display will differ based on this class. You can leave it blank to display a normal button.

- *$level 3*: level, used to group buttons together and related to parameter 4 "sort_order"

- *$sortOrder:* sort_order, used to arrange the orders of buttons of the same level.

- *$area*: area gets two values "header" and "footer". The default value is header, but if you want to display button in the footer, you need to choose "footer".

3.2. Deleting custom buttons

To delete custom buttons, you can simply delete the code used to add that button.

Follow the link

app/code/local/{your_namespace}/{your_module}/block/adminhtml/{section}/

Go to file **Edit.php**

In the function __construct() of the file Edit.php and add this code:

```
public function __construct()
```

```
        {
            parent::__construct();
            $this->_removeButton('delete');

        }
```

The Function _removeButton() has a parameter which is the ID of button. If you want to delete any button, you can use the function removeButton. I advise you to follow this link to go to file Container.php to understand more clearly:

app/code/core/Mage/Adminhtml/Widget/Container/app/code/core/Mage/Adminhtml/Widget/Form/Container/

Because the base form of this function is quite simple, you can learn it by yourself.

3.3. Updating custom buttons

Similar to deleting a button, Magento supports editing attributes of an available button so that you can control the number of buttons in form.

Steps are similar to those when you delete a button. A difference is that I added the following code in the function __construct():

```
public function __construct()
        {
            parent::__construct();

            $this->_updateButton('save', 'label',
Mage::helper('lesson10')->__('Apply'));

        }
```

In the above code, I have used the function _updateButton to change the label of the button "Save" in form to "Apply".

Explanation of this function:

Method prototype:

```
protected function _updateButton($id, $key=null, $data){}
```

- *$id:* button's id

- *$key:* The properties of the button

- *$data:* The value of the properties after update

III.SUMMARY

In lesson 10, we have learned to manage Magento backend forms:

- Add normal HTML tabs vs. AJAX-based tabs

- Render Magento normal fields vs. custom fields
- Add, edit or delete buttons.

Back to top

Lesson 11: Magento Configuration

I. INTRODUCTION

In this lesson, we will study how Magento Multiple Stores system in Magento works, the installation and configuration of a Magento website as well as the use of these configuration values in stores.

1. Time: 2 hours

2. Content

- Magento Multiple Stores system in Magento
- Adding configurations to a Magento system
- Configuring default values from code
- How to use/get configuration values from the system

II. CONTENT

1. Magento Multiple Stores system in Magento

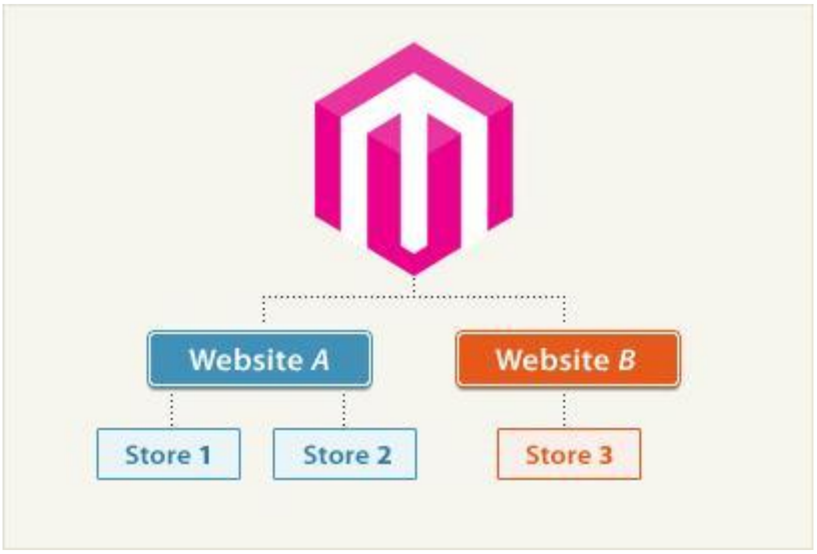
One of Magento advanced features is that it allows us to manage multiple websites and stores within one installation, and we have an amazing system to support this: GWS – aka "Global, Website and Store."

- Global: This refers to the entire installation.

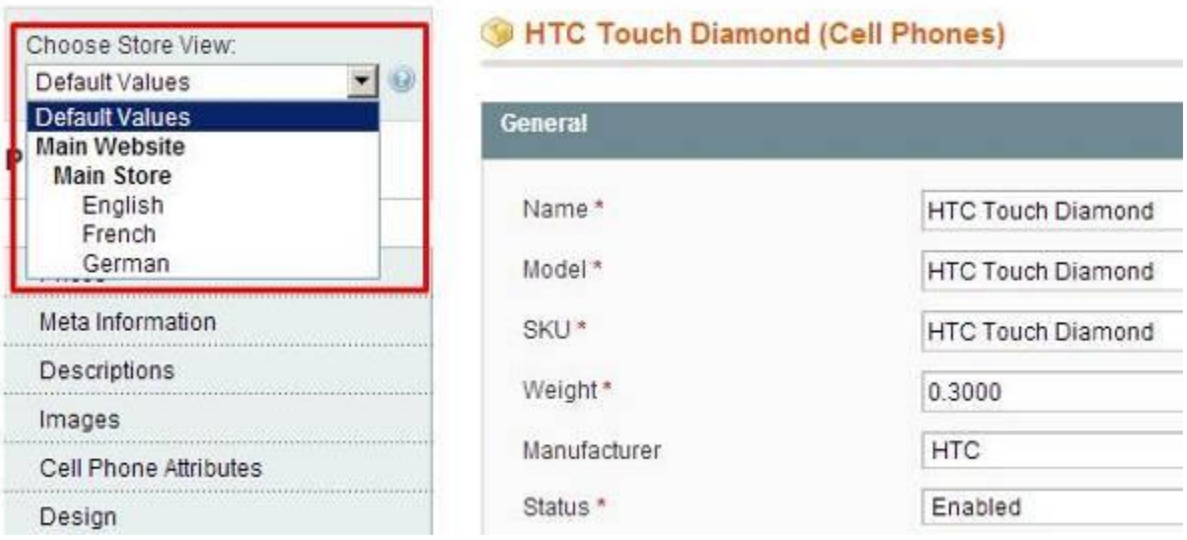
- Website: Websites are 'parents' of stores. A website consists of one or more stores. Websites can be set up to share customer data, or not to share any data.

- Store (or store view group): Stores are 'children' of websites. Products and Categories are managed at the store level. A root category is configured for each store view group and allows multiple stores under the same website to have totally different catalog structures.

- Store View: A store needs one or more store views to be browse-able in the frontend. The catalog structure per store view is always similar, it simply allows for multiple presentations of the data in the front. 90% of implementations will likely use store views to allow customers to switch between two or more languages.



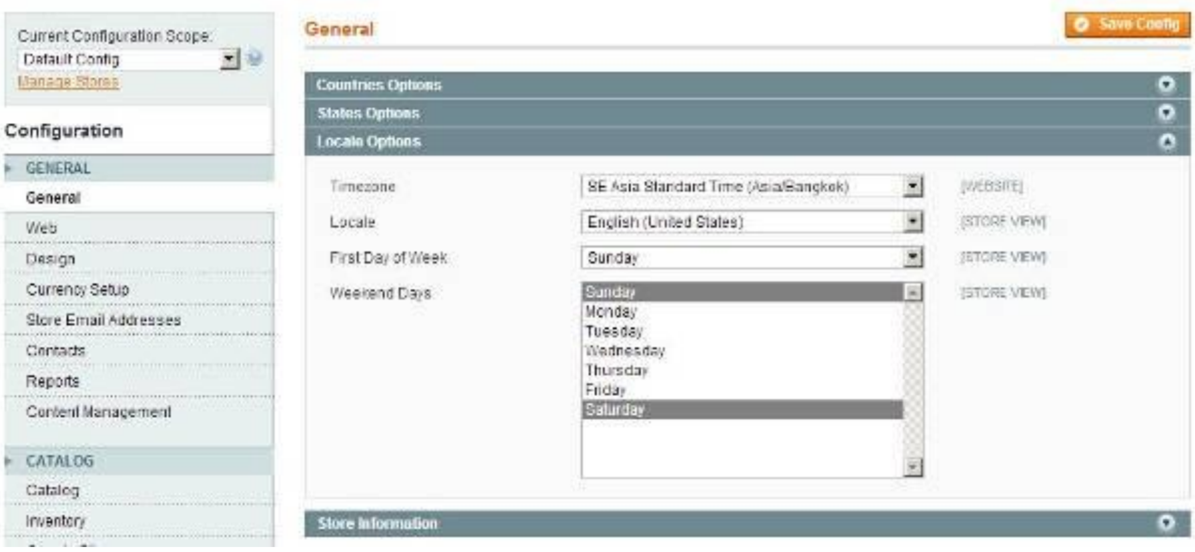
When managing data in backend, we have a dropdown box to switch between store views.



Name *	HTC Touch Diamond
Model *	HTC Touch Diamond
SKU *	HTC Touch Diamond
Weight *	0.3000
Manufacturer	HTC
Status *	Enabled

2. Adding configurations to Magento system

Magento is an exclusive configuration system for all modules running on it. Administrators do these configurations by accessing **System => Configuration**. On this page, we will view a configuration form, with a dropdown box to switch between websites and store views.



Timezone: SE Asia Standard Time (Asia/Bangkok) [WEBSITE]
Locale: English (United States) [STORE VIEW]
First Day of Week: Sunday [STORE VIEW]
Weekend Days: [✓ Sunday] Monday Tuesday Wednesday Thursday Friday [✓ Saturday] [STORE VIEW]

Magento get these system configuration forms from **system.xml** files in etc folers of all working modules. As a result, to add configurations to the Magento system, we need to add definition codes to this file.

First, create a module named **Magestore_Lesson11** and edit file **system.xml** in the folder file **app/code/local/Magestore/Lesson11/etc.**

Add a Tab to the left navigation in the configuration list of Magento system.

```
<config>
    <tabs>
        <magestore translate="label">
            <label>Magestore Extension</label>
            <sort_order>400</sort_order>
        </magestore>
    </tabs>
</config>
```

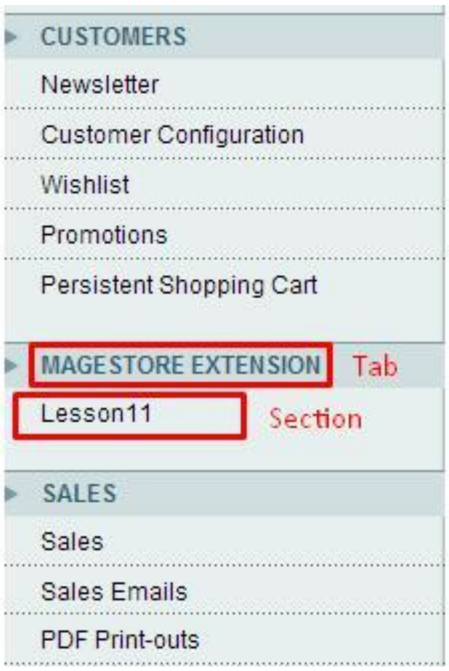
- ✓ magestore: tab identify
- ✓ label: tab name
- ✓ sort_order: tab display order among all the tab of Magento

Then we will create a section for module configuration. We will add this piece of code:

```
<config>
    <tabs>
        ...
    </tabs>
    <sections>
        <lesson11 translate="label" module="lesson11">
            <class>separator-top</class>
            <label>Lesson11</label>
            <tab>magestore</tab>
            <frontend_type>text</frontend_type>
            <sort_order>299</sort_order>
            <show_in_default>1</show_in_default>
            <show_in_website>1</show_in_website>
            <show_in_store>1</show_in_store>
        </lesson11>
    </sections>
</config>
```

- ✓ **lesson11**: section identify
- ✓ **class**: html class for the link to the section
- ✓ **label**: screen name of section lesson11
- ✓ **tab**: tab identify in which the section lies. In this example, section lesson11 lies in tab Magestore
- ✓ **frontend_type**: in Magento version 1.7.0.2, we can go without declaring this tag
- ✓ **sort_order**: the order display for this section, this matters when our tab has more than 1 section
- ✓ **show_in_default**: display this section in 'default' scope, i.e. the Global level stated above
- ✓ **show_in_website**: display this section in 'website' scope. This matters when we reselect configuration scope (by switching from the dropdown) as website.
- ✓ **show_in_store**: display this section in 'store view' scope.

Now in configuration system, we can see:



Next, we will add configurations to the elements of the configuration form:

```
<config>
    <tabs>
        ...
    </tabs>
    <sections>
        <lesson11 translate="label" module="lesson11">
            ...
            <groups>
                <general translate="label">
                    ...
                    <fields>
                        <enable translate="label">
                            <label>Enable</label>
                            <frontend_type>select</frontend_type>
                            <sort_order>1</sort_order>

<source_model>adminhtml/system_config_source_yesno</source_model>
                            <show_in_default>1</show_in_default>
                            <show_in_website>1</show_in_website>
                            <show_in_store>1</show_in_store>
                            <comment></comment>
                        </enable>
                    </fields>
                </general>
            </groups>
        </lesson11>
    </sections>
</config>
```

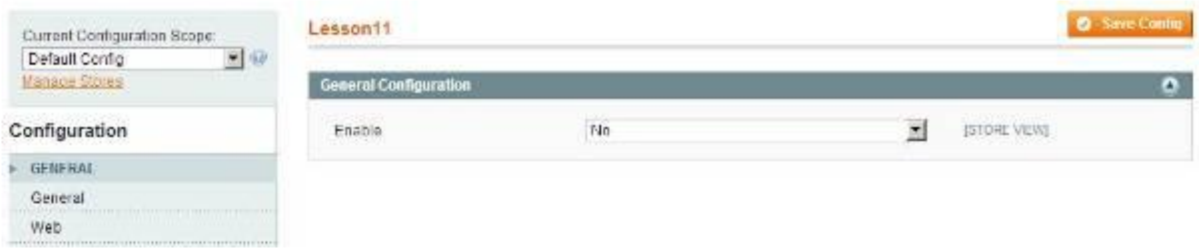
We should pay attention to the following tab:

- ✓ **enable**: code of the field we will add
- ✓ **frontend_type**: the display type of the field configured on the form. We have some types such as select, multiselect, text, textarea, editor, file, image, label, radio, hidden. It is like when we create a field in a backend form (in Lesson 10).
- ✓ **source_model**: the model used to provide data to a select or multiselect when we display on the form. In this example, we can see the file source model like this:

```
class Mage_Adminhtml_Model_System_Config_Source_Yesno
{
    /**
     * Options getter
     *
     * @return array
     */
    public function toOptionArray()
    {
        return array(
            array('value' => 1, 'label'=>Mage::helper('adminhtml')-
>__('Yes')),
            array('value' => 0, 'label'=>Mage::helper('adminhtml')-
>__('No')),
            );
        }
}
```

- ✓ **comment**: display notes right below the configurations we have added. Besides, we can also declare other tags (if not, default values will be added).
- ✓ **frontend_model**: the block to show the HTML code piece for field, it seems a rederer for this field.
- ✓ **backend_model**: the model used to customize and change data when we save to database or get out data configured from database.
- ✓ **depends**: this field allows configuring whether to display it or not depends on another field value.

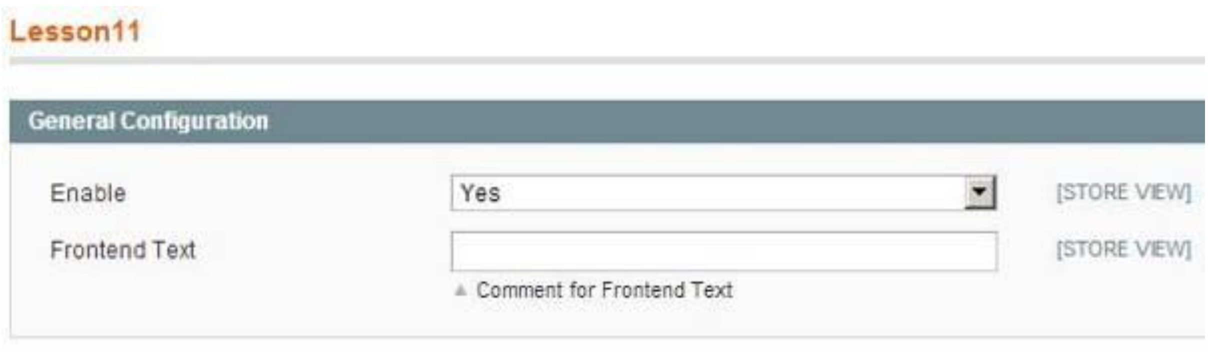
We will get a configuration form like this:



Or adding another field depends on this ‘enable’ field, we add these codes:

```
<fields>
    <enable translate="label">
        ...
    </enable>
    <frontend_text translate="label">
        <label>Frontend Text</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>
        <comment>Comment for Frontend Text</comment>
        <depends>
            <enable>1</enable>
        </depends>
    </frontend_text>
</fields>
```

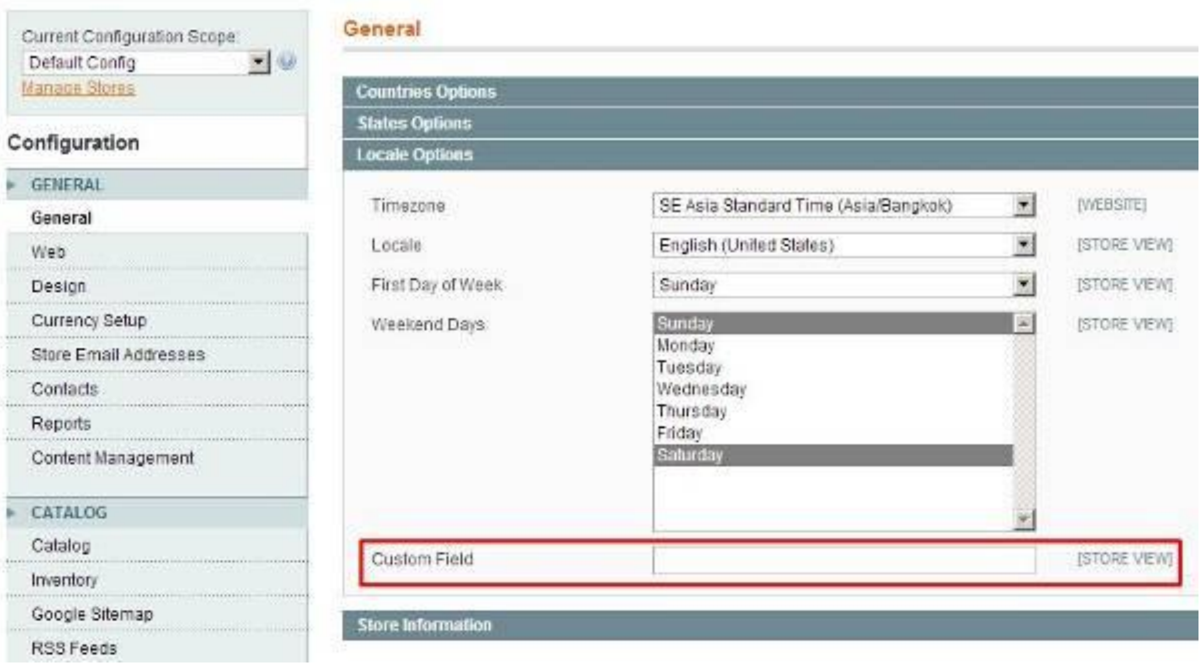
And get this configuration form:



Moreover, we can add fields to configuration forms available in Magento by clarifying the path to the section or group of that form.

```
<config>
    ...
    <sections>
        <lesson11 translate="label" module="lesson11">
            ...
        </lesson11>
        <general>
            <groups>
                <locale>
                    <fields>
                        <custom_field translate="label">
                            <label>Custom Field</label>
                            <frontend_type>text</frontend_type>
                            <sort_order>100</sort_order>
                            <show_in_default>1</show_in_default>
                            <show_in_website>1</show_in_website>
                            <show_in_store>1</show_in_store>
                        </custom_field>
                    </fields>
                </locale>
            </groups>
        </general>
    </sections>
</config>
```

Here is what appears in the configuration form:



Timezone	SE Asia Standard Time (Asia/Bangkok)	[WEBSITE]
Locale	English (United States)	[STORE VIEW]
First Day of Week	Sunday	[STORE VIEW]
Weekend Days	Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday	[STORE VIEW]
Custom Field		[STORE VIEW]

3. Configuring Magento default values from code

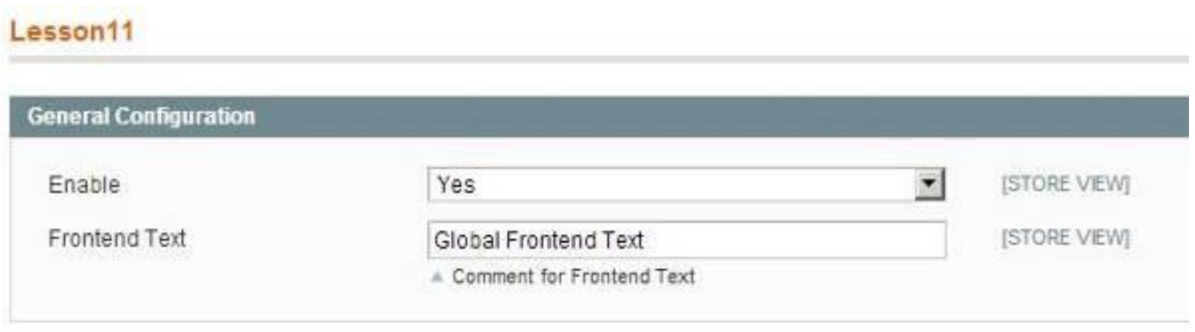
In the above configuration forms, we have created the fields for admin to configure their values. However, if he has not configured yet, we need Magento default values so that the modules can work well for our programming purposes.

To configure these Magento default values, we put some code into the file **config.xml** in the same directory with the file **system.xml**. Add the code like this:

```
<config>
    ...
    <default>
        <lesson11>
            <general>
                <enable>1</enable>
                <frontend_text>Global Frontend Text</frontend_text>
            </general>
        </lesson11>
    </default>
</config>
```

- ✓ **default**: the value scope of the configuration. '**default**' is equivalent to Global. Besides we can use other tags such as websites and stores, with which we need more information about the website (website code) or the store (store code)
- ✓ **lesson11**: the section code in the above configuration
- ✓ **general**: the code group in the above configuration
- ✓ **enable**: the code field in the above configuration

The configuration form will look like this:


Lesson11

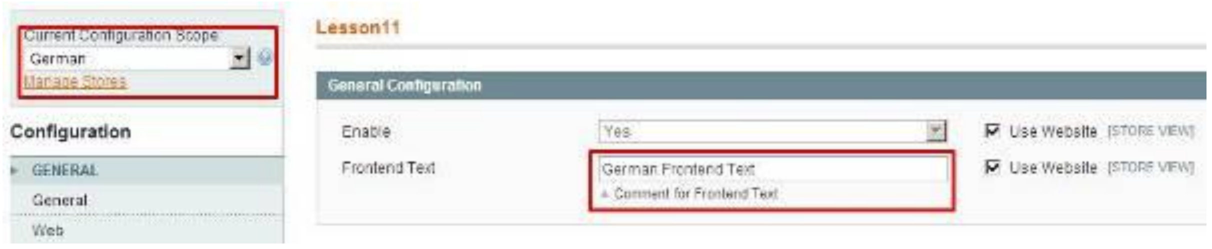
General Configuration

Enable	Yes	[STORE VIEW]
Frontend Text	Global Frontend Text Comment for Frontend Text	[STORE VIEW]

Or add default configuration for the store view ‘German’ only:

```
<config>
    ...
    <default>
        <lesson11>
            <general>
                <enable>1</enable>
                <frontend_text>Global Frontend Text</frontend_text>
            </general>
        </lesson11>
    </default>
    <stores>
        <german>
            <lesson11>
                <general>
                    <frontend_text>German Frontend Text</frontend_text>
                </general>
            </lesson11>
        </german>
    </stores>
</config>
```

Then we will have the default configuration for the store view ‘German’ only.


Current Configuration Scope: German

Manage Stores

Configuration

GENERAL

General

Web

Lesson11

General Configuration

Enable	Yes	☑ Use Website	[STORE VIEW]
Frontend Text	German Frontend Text Comment for Frontend Text	☑ Use Website	[STORE VIEW]

4. How to use/get configuration values from the system

With the above configuration, we will use the value configured by Admin for our activities in our module.

To get the configuration values, we will use the following method

```
Mage::getStoreConfig($path, $store = null)
```

- ✓ **path**: the path to the configuration, in form of [section_code]/[group_code]/[field_code]
- ✓ **store**: the store we want to get the configuration from. If it is null, the Magento system will get the configuration value from the current store on which the system is running.

For example, we call the function:

```
Mage::getStoreConfig('lesson11/general/frontend_text');
// Result: 'Global Frontend Text'
```

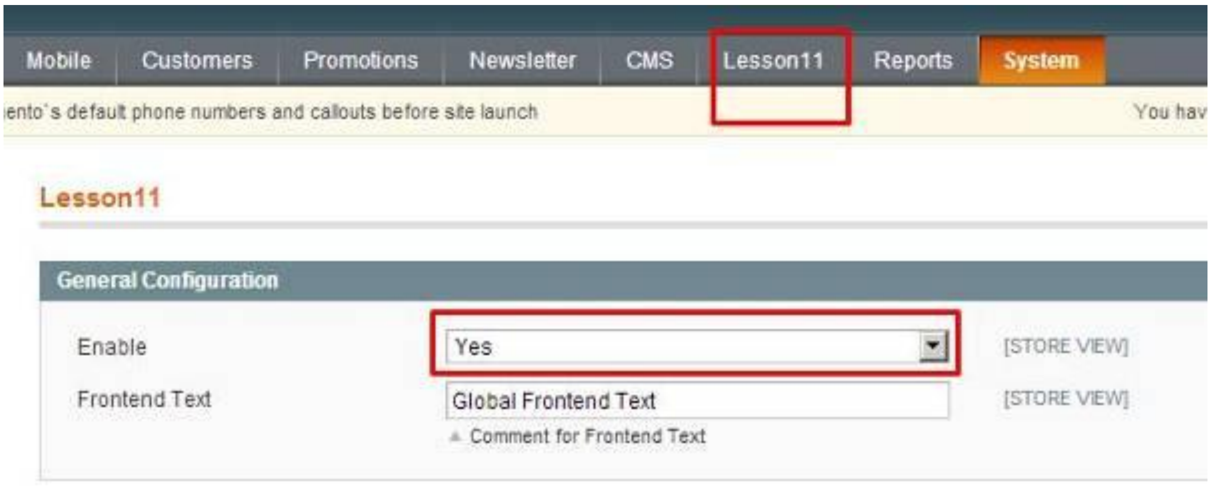
Besides, if the path is not full to **field_code**, the returned value will be a value array of the configuration field.

```
Mage::getStoreConfig('lesson11/general');
// Array(
//       'enable' => 1
//       'frontend_text' => 'Global Frontend Text'
// )
```

Moreover, we can also use the configurations to trigger the menu display in backend or to perform an action from the file layout of the template. We can add it into the file **adminhtml.xml** in the same directory with the file **system.xml** like this:

```
<config>
    <menu>
        <lesson11 module="lesson11" translate="title">
            <title>Lesson11</title>
            <sort_order>71</sort_order>
            <depends>
                <config>lesson11/general/enable</config>
            </depends>
            <children>
                ...
            </children>
        </lesson11>
    </menu>
    ...
</config>
```

We declare in depends/config tag. As a result, when we configure Enable as 'Yes', a menu will be displayed like this:



Lesson11

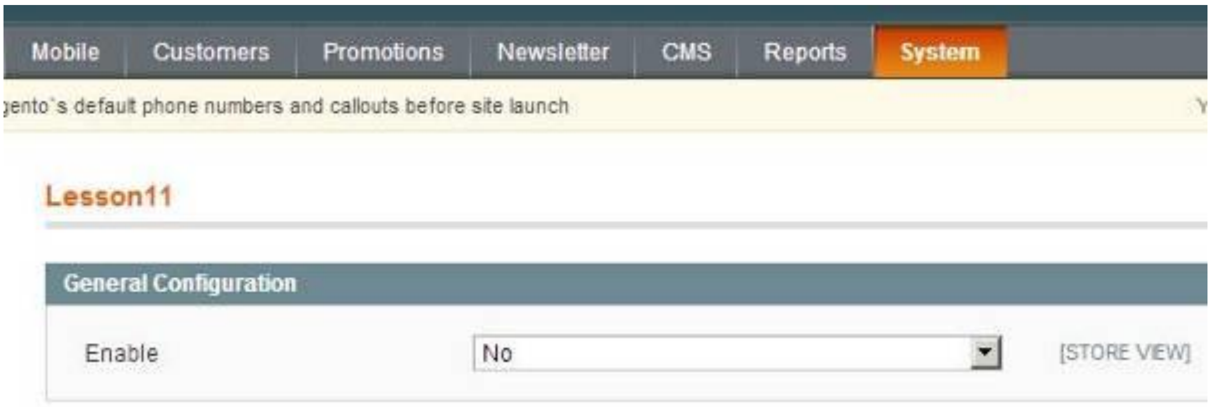
General Configuration

Enable: Yes [STORE VIEW]

Frontend Text: Global Frontend Text [STORE VIEW]

Comment for Frontend Text

If we configure Enable as ‘No’, it will no longer be displayed in the menu bar.


Mobile
Customers
Promotions
Newsletter
CMS
Reports
System
gento's default phone numbers and callouts before site launch

Lesson11

General Configuration

Enable: No [STORE VIEW]

III. SUMMARY

In this lesson, we have learned how to work with the configuration system in Magento including:

- Adding, configuring values from codes for store
- Using configuration values for programming

We need this topic in almost any module development as it enables creating different options for module users.

Back to top

Lesson 12: Magento Layout & Template in Frontend

I. INTRODUCTION

1. Time: 3 hours

2. Content:

- Creating a simple Magento template in frontend
- Configuring Magento template packages
- Working with the layout file

II. CONTENT

1. Creating a simple Magento template in frontend

To create a simple template on frontend we need some elements:

- Controller/Action: the method to process a request from a corresponding URL.
- Block files: to control template blocks, containing the methods which can be called directly in the template it controls.
- Template files: to contain the HTML code
- Layout files: to configure controller, block and template.

Process to create a simple template on frontend of Magento, the illustrative example is written in module Lesson 12:

1.1. Creating controller/action

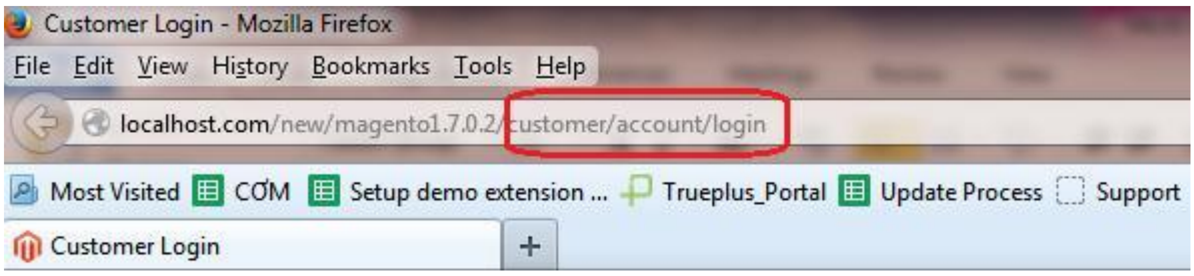
A URL in Magento browser looks like this:

http://base_path/routers_name/controller_name/action_name/param/value

In which:

- base_path: the website domain name
- routers_name: the frontend URL of a module configured in the file Name_Space/Module/etc/config.xml
- controller_name: the file name (class) in the folder: Name_Space/Module/controllers

- action_name: the method name in the above class in the form of "action_nameAction ()"
  param & value: the associated parameters and values


Customer Login - Mozilla Firefox
File Edit View History Bookmarks Tools Help
localhost.com/new/magento1.7.0.2/customer/account/login
Most Visited COM Setup demo extension ... Trueplus_Portal Update Process Support
Customer Login

Example of controller customer/account/login

There are three steps to create a controller/action in Magento

Step 1: Configure routers_name in file config.xml:

app/code/local/Magestore/Lesson12/etc/config.xml

```
<config>
   …
   <frontend>
        <routers>
            <lesson12>
                <use>standard</use>
                <args>
                    <module>Magestore_Lesson12</module>
                    <frontName>lesson12</frontName>
                </args>
            </lesson12>
        </routers>

    </frontend>
    …
</config>
```

In which:

- <routers>, <args>: config tags
- <lesson12>: module name
- Magestore_Lesson12: NameSpace_ModuleName
- <frontName>: routers_name

Step 2: Create file controller. This file is a class used to extend

Mage_Core_Controller_Front_Action in the folder:

app/code/local/Magestore/Lesson12/controllers/IndexController.php

Step 3: Create Action which means a method in the above class, for example: helloAction().

```
class Magestore_Lesson12_IndexController extends
Mage_Core_Controller_Front_Action
{
    /**
     * hello action
     */
    public function helloAction()
    {
        $this->loadLayout();
        $this->renderLayout();
    }
}
```

1.2. Creating a layout file:

Layout file is a XML file which can configure Controllers/Action with associated block and template. This file is put in folder app/design/frontend/package/theme/layout (package and theme are names of Magento templates.

For example: To create file layout for module Lesson 12

app/design/frontend/default/default/layout/lesson12.xml

```
<layout version="0.1.0">
    <default>
        <!-- update layout for all frontend page -->
    </default>
    <lesson12_index_hello>
        <reference name="content">
            <block type="lesson12/lesson12" name="lesson12"
template="lesson12/lesson12.phtml" />
        </reference>
    </lesson12_index_hello>
</layout>
```

In the above example:

- ✓ **<lesson12_index_hello>** is the name of Controller/Action that needs to be configured; this controller belongs to module *Lesson 12* in class *Indexcontroller.php* and method *helloAction*.
- ✓ **<reference name="content">** is the name of block we want to do refer. Their names are written in the root configuration of Magento. The declaration of block in this tag will be the child block of block content when rendering to HTML (app/design/frontend/base/default/layout/page.xml).
- ✓ **type**: name of block in the right URL of folder. The file above is *Lesson.php* in the folder *app/code/local/Magestore/Lesson/Block/Lesson12.php*.

- ✓ **name**: The name of block we set.
- ✓ **template**: We name template file with the right URL after configuring template packages in admin. In this example, we implement in the default template of Magento app/design/frontend/default/default/template/lesson12/lesson12.phtml

1.3. Creating a block file:

Declare the module's block in file config.xml

```
<config>
    <global>
        …
        <blocks>
            <lesson12>
               <class>Magestore_Lesson12_Block</class>
            </lesson12>
        </blocks>
        …
   </global>
</config>
```

Create file *app/code/local/Magestore/Lesson12/Block/Lesson12.php* with a simple method *getHelloString()*

Note: The class in the block is named precisely according to the folder path

```
class Magestore_Lesson12_Block_Lesson12 extends Mage_Core_Block_Template
{
    public function getHelloString()
    {
        return 'Welcome to Lesson 12';
    }
}
```

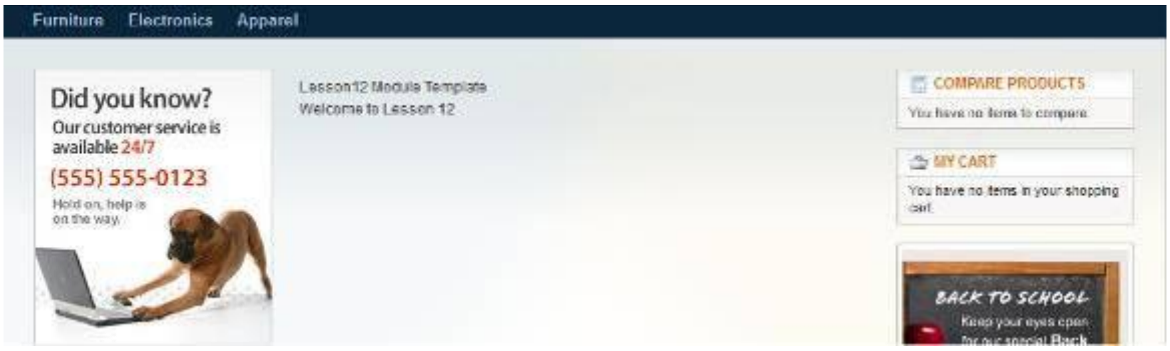
1.4. Creating a template file:

File template has the extension .phtml which can contain HTML, PHP, CSS, JAVASCRIPT code. In this example, we create a template file with a simple HTML and PHP code such as: *app/design/frontend/default/default/templste/lesson12/lesson12.phtml*

```
<div class="lesson">
    <?php echo $this->__('Lesson Module Template') ?>
    <br/>
    <?php echo $this->getHelloString() ?>
</div>
```

Function *getHelloString* is generated corresponding to block of Lesson 12 using object *$this.*

Note: The characters want to be shown on frontend should be called out by PHP command, for example, $this->__('Lesson Module Template') This enables using Magento language interpreting system. After finishing these steps, we can save the direct link and test in the browser: http://yoursite.com/lesson12/index/hello


Furniture
Electronics
Apparel
Did you know?
Our customer service is available 24/7
(555) 555-0123
Hold on, help is on the way.
COMPARE PRODUCTS
MY CART
BACK TO SCHOOL
Keep your eyes open for our special Back

Lesson12 Module Template
Welcome to Lesson 12

You have no items to compare.

You have no items in your shopping cart.

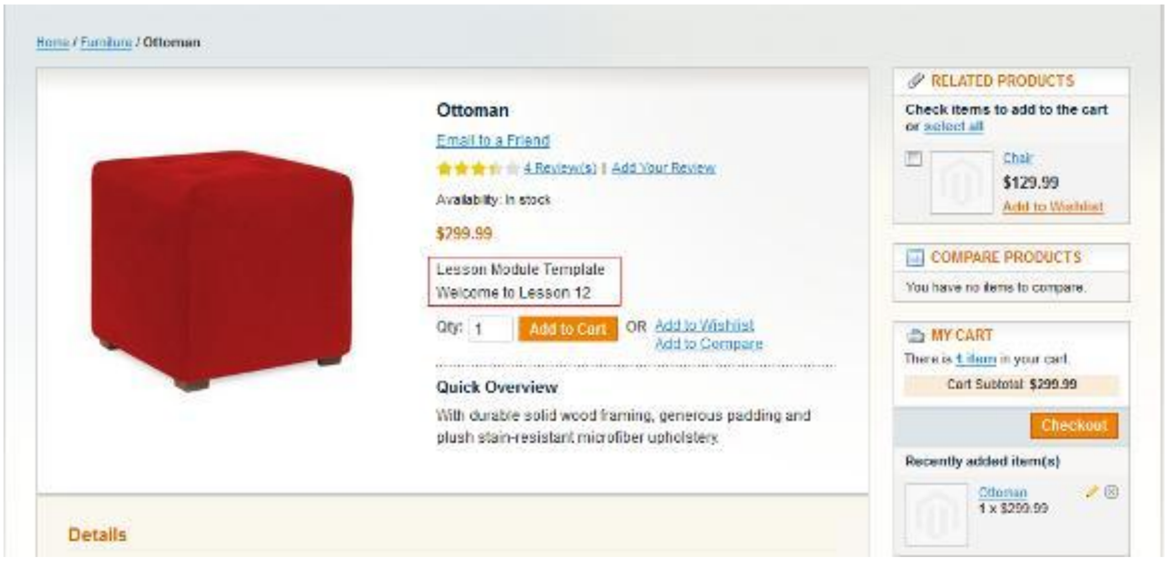
If you want to insert template into an existing page for example product pages without any intervention to Magento module core, we follow these steps:

Firstly, go to the Magento layout core file see if the blocks have been written by this modules: *app/design/frontend/base/default/layout/catalog.xml*. We can easily notice that there is block *"product.info.extrahint"* in the tag <*catalog_product_view*> and we will make reference to this block.

Secondly, configure file layout:

```
    <catalog_product_view>
        <reference name="product.info.extrahint">
               <block type="lesson12/lesson12" name="lesson12"
template="lesson12/lesson12.phtml" />
        </reference>
    </catalog_product_view>
```

This example reuses the results of block and template in the above part. The page will look like this:



Availability: In stock

$299.99

Lesson Module Template
Welcome to Lesson 12

With durable solid wood framing, generous padding and plush stain-resistant microfiber upholstery.

Check items to add to the cart or select all

You have no items to compare.

There is 1 item in your cart.

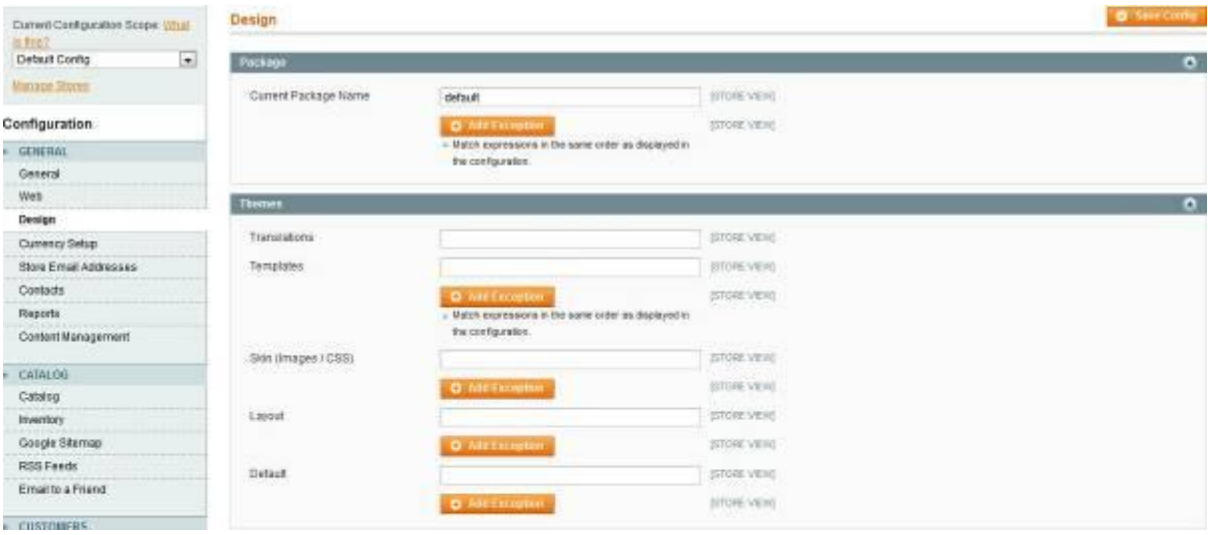
Cart Subtotal: $299.99

2. Configuring Magento template package in frontend

2.1. Steps to configure template package

To begin, you follow the path System -> Configuration -> Design. The right side is the Package and Theme tab. These tabs deal with the configuration tasks on frontend. Package is the name of folder app/design/frontend; theme is the folder containing template files which is located in the package folder.

When we create a module, information in these tabs is usually left blank, which means website uses the default template: package -> default; theme -> default.

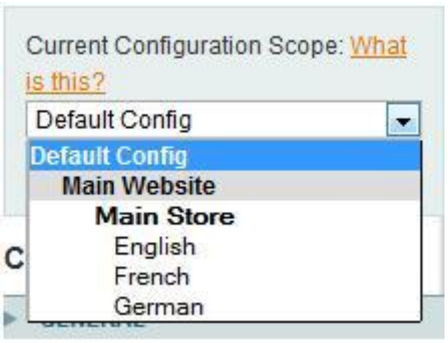


Current Package Name: default

Match expressions in the same order as displayed in the configuration.

Translations:
Templates:
Skin (Images / CSS):
Layout:
Default:

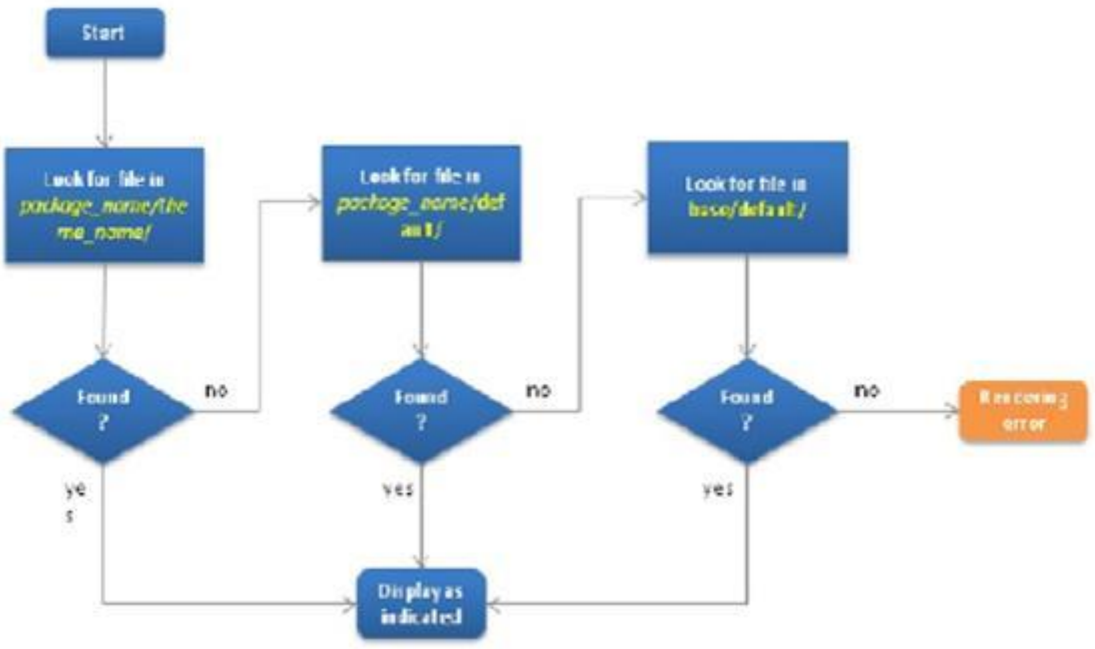
Besides, in Magento, we can use multi-store management to change the template of each individual store: Use select box on the top left to select the store you want to change, and then fill in the package and theme corresponding to that store.



2.2. Magento Fall - Back Design

Magento Design Fall-Back is used to render themes from the Magento basic templates. The fall-back hierarchy is described from Custom_theme -> default_theme -> base -> error.

Look at the following diagram:


Start
Look for file in package_name/theme_name/
Look for file in package_name/default/
Look for file in base/default/
Found ?
no
yes
Rendering error
Display as indicated

Design Fall-Back process has four steps:

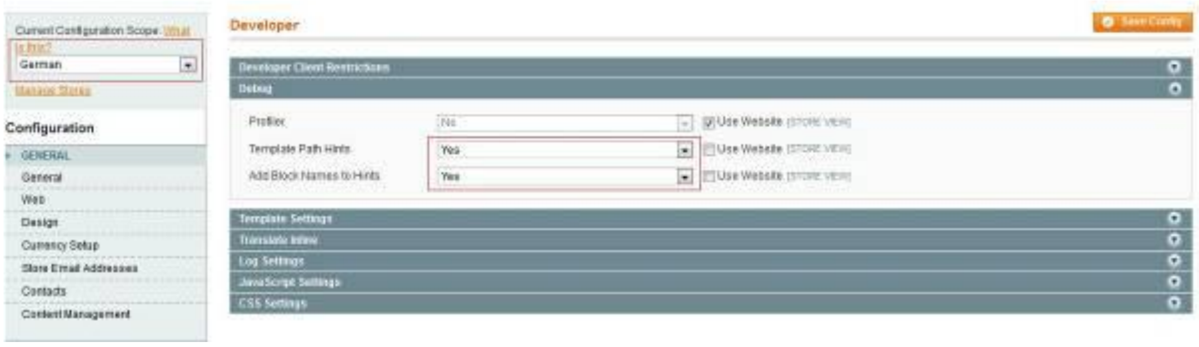
- Search the request file in the folder:
    - app/design/frontend/custom_package/custom_theme/
    - skin/frontend/custom_ package/custom_theme
- If fail to find it, search in the folder:
    - app/design/frontend/custom_package/default
    - skin/frontend/custom_package/default
- In case you are unable to look for the request file, continue searching in the folder
    - app/design/frontend/base/default
    - skin/frontend/base/default
- The error notification will be displayed if the file cannot be found.

2.3. Turning on template_path_hint

For a Magento programmer or a Magento starter, there is a way to show which block and template is displayed in a page. You can use “template_path_hint” funtion by a few minor tasks in admin as follows:

From admin page, select *System* -> *Configuration* -> *Developer*

In this page, we show store to display template_path_hint as above suggestion. In the right content, we choose "Debug" tab and then we change the value of two selected boxes: "Template Path Hint" and "Add Block Name to Hints" with Yes selection. Finally, choose "Save Config"


Current Configuration Scope: German

Developer

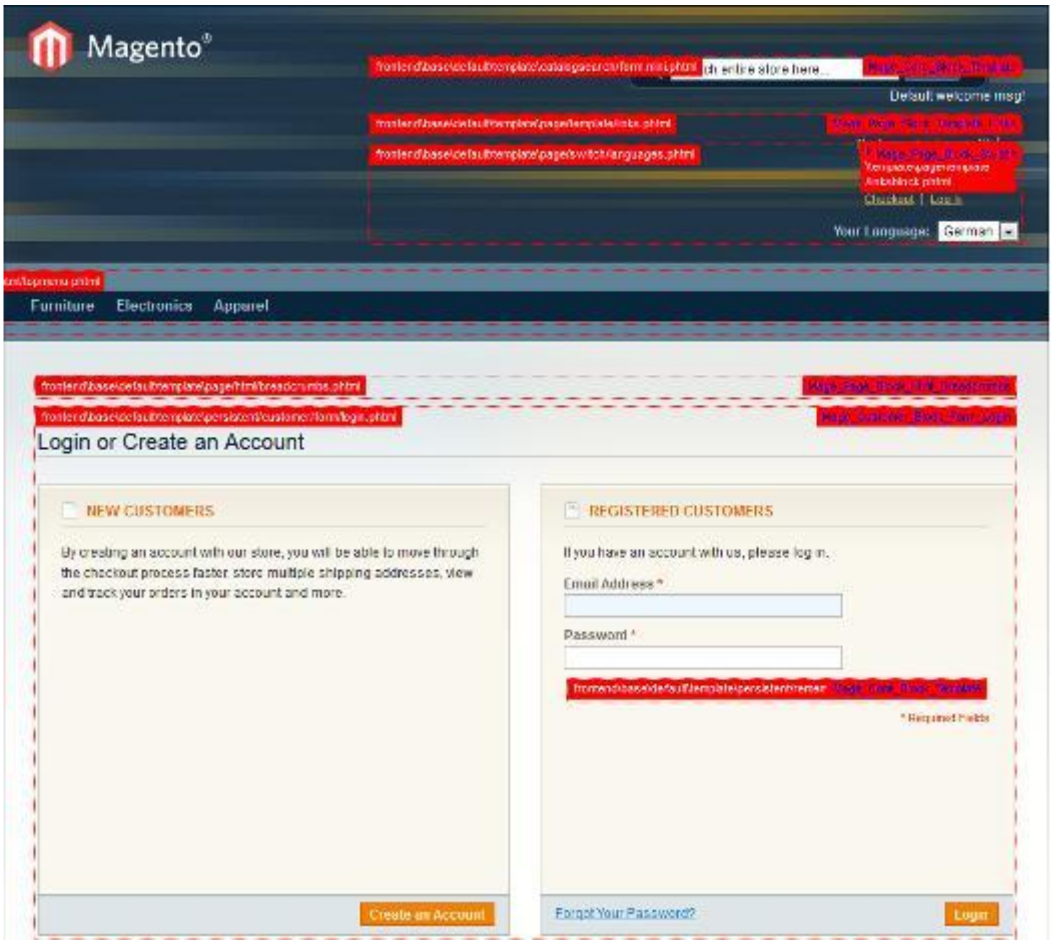
Debug

Profiler: No — ✓ Use Website [STORE VIEW]

Template Path Hints: Yes — Use Website [STORE VIEW]

Add Block Names to Hints: Yes — Use Website [STORE VIEW]

Result: (Noted that selected store is German)


frontend/base/default/template/page/html/breadcrumbs.phtml

frontend/base/default/template/persistent/customer/form/login.phtml

Login or Create an Account

NEW CUSTOMERS

By creating an account with our store, you will be able to move through the checkout process faster, store multiple shipping addresses, view and track your orders in your account and more.

REGISTERED CUSTOMERS

If you have an account with us, please log in.

Email Address *

Password *

* Required Fields

3. Working with file layout in Magento

In the configuration of file layout, in addition to the usage of block and template to create Magento frontend, we can use available functions to insert file or the essential path such as CSS file, JavaScript, TopLink and footerlink

For example, app/design/frontend/default/default/layout/lesson12.xml

```
<lesson12_index_hello>
        <reference name="head">
            <action method="addCss">
                
<stylesheet>css/magestore/lesson12/lesson12.css</stylesheet>
            </action>
            <action method="addJs">
                <script>magestore/lesson12/lesson12.js</script>
            </action>
        </reference>
        <reference name="content">
            <block type="lesson12/lesson12" name="lesson"
template="lesson12/lesson12.phtml" />
        </reference>
</lesson12_index_hello>
```

The methods in <action>tag are written in file

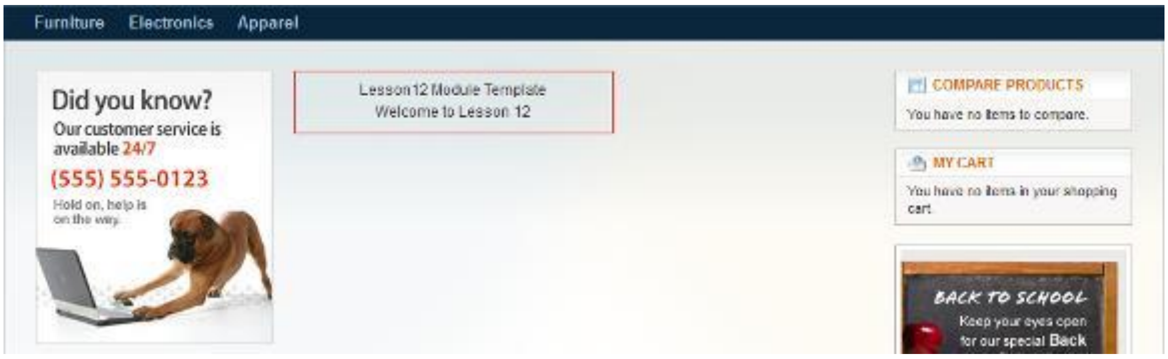
app/code/core/Mage/Page/Block/Html/Head.php through file layout page.xml

In the above example, you can insert more CSS request into file lesson12.css, which is located in the folder:

skin/frontend/default/default/css/magestore/lesson12/lesson12.css

```
.lesson12{
    border: 1px solid red;
    height: 40px;
    padding: 5px;
    text-align: center;
    width: 150px;
}
```

The result will be:


Furniture
Electronics
Apparel
Did you know?
Our customer service is available 24/7
(555) 555-0123
Hold on, help is on the way.
COMPARE PRODUCTS
You have no items to compare.
MY CART
You have no items in your shopping cart.
BACK TO SCHOOL
Keep your eyes open for our special Back

Lesson12 Module Template
Welcome to Lesson 12

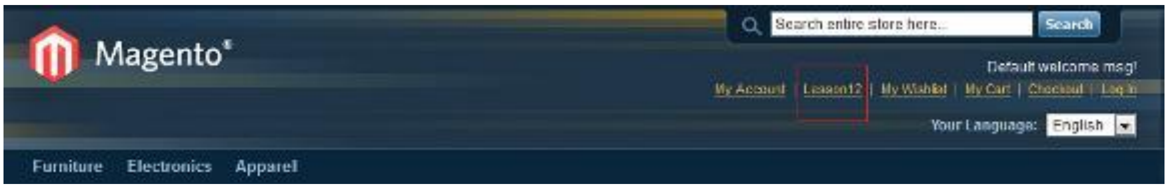
Another example of using addLink function to add a link on the first page of Magento: this function is written in the Block: app/code/core/Mage/Page/Block/Htm/Toplinks.php

```
  <default>
      <reference name="top.links">
          <action method="addLink" translate="label title" 
module="lesson12">
              <label>Lesson12</label>
              <url>lesson12/index/hello</url>
              <title>Lesson12</title><prepare/><urlParams/>
              <position>10</position>
          </action>
      </reference>
  </default>
```

In this example the function addLink is written in <default> tag aiming to be implemented in all pages of a website. The attributes of this function contain:

- Label: Label displayed link
- URL: The path to controller/Action
- Title: The title displayed on the browser tab
- Position: Link position compared with the pre-existing link

The result will be displayed:


Magento
Search entire store here...
Search
Default welcome msg!
My Account | Lesson12 | My Wishlist | My Cart | Checkout | Log In
Your Language: English
Furniture
Electronics
Apparel

III. SUMMARY AND REFERENCES

After this lesson, we should master the following knowledge:

- How to create controller/Action in Magento
- How to configure the template package in admin
- How to create a simple template on frontend
- How to use some basic functions in xml

And you should read these references:

- Lesson 11: Magento Configuration

- Blanka, 2012. *Topic 3 - Part 3: Template Structure - Fallbacks*. Available at http://blog.magestore.com/2012/02/08/magento-certificate-template-structure-fallbacks/
- Magentocommerce, n.d.. *Magento Design Terminologies*. Available at: http://www.magentocommerce.com/design_guide/articles/magento-design-terminologies4

Back to top

Lesson 13: Data Grid in Frontend

I. INTRODUCTION

1. Time: 3 hours

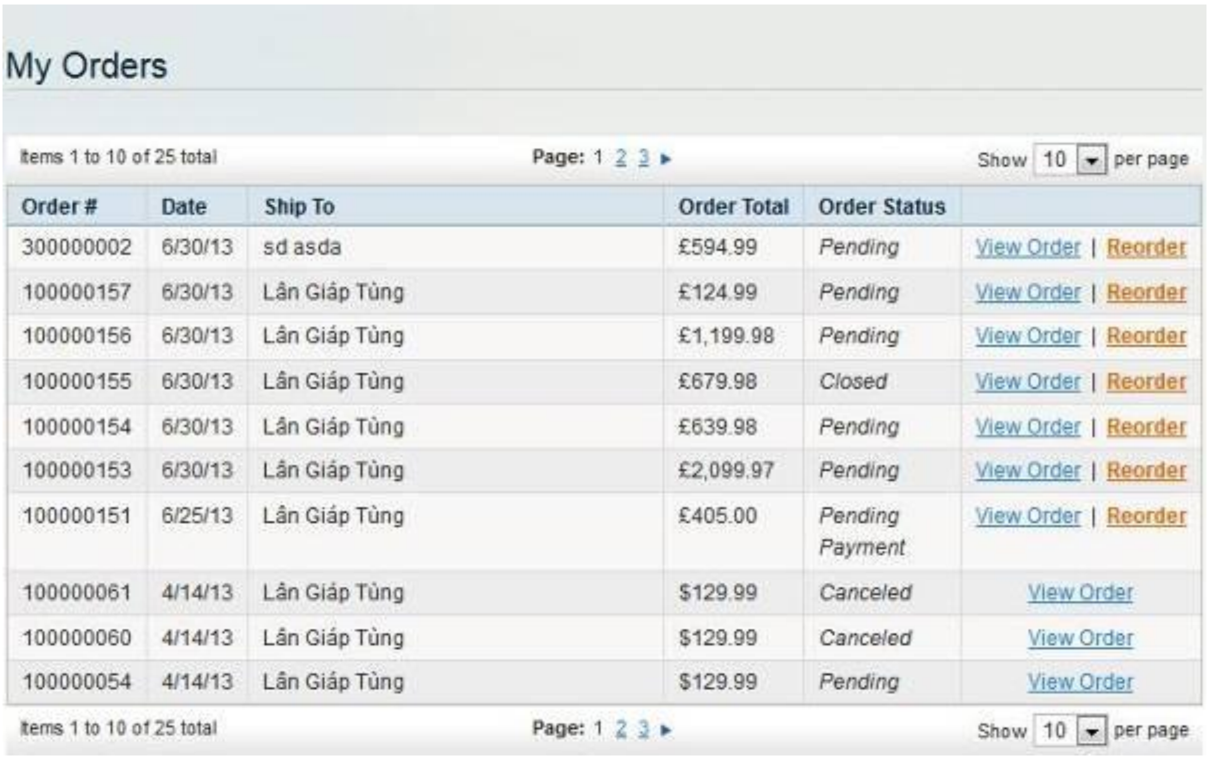
2. Content

- Introduction to Grid Magento
- Process to create a Magento grid
- Writing Grid template
- Grid pagination

II. CONTENT

1. Introduction to Magento Grid

Here is an example of grid, which is a page displaying a list of orders when you sign in a Magento account



My Orders

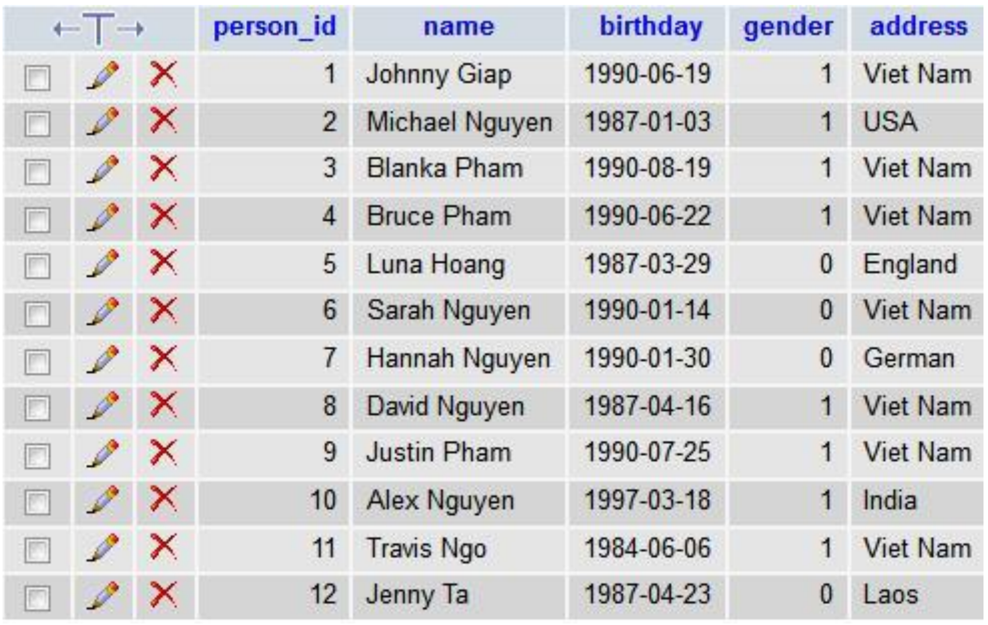
Items 1 to 10 of 25 total — Page: 1 2 3 ▸ — Show 10 per page

Order #	Date	Ship To	Order Total	Order Status	
300000002	6/30/13	sd asda	£594.99	Pending	View Order | Reorder
100000157	6/30/13	Lân Giáp Tùng	£124.99	Pending	View Order | Reorder
100000156	6/30/13	Lân Giáp Tùng	£1,199.98	Pending	View Order | Reorder
100000155	6/30/13	Lân Giáp Tùng	£679.98	Closed	View Order | Reorder
100000154	6/30/13	Lân Giáp Tùng	£639.98	Pending	View Order | Reorder
100000153	6/30/13	Lân Giáp Tùng	£2,099.97	Pending	View Order | Reorder
100000151	6/25/13	Lân Giáp Tùng	£405.00	Pending Payment	View Order | Reorder
100000061	4/14/13	Lân Giáp Tùng	$129.99	Canceled	View Order
100000060	4/14/13	Lân Giáp Tùng	$129.99	Canceled	View Order
100000054	4/14/13	Lân Giáp Tùng	$129.99	Pending	View Order

Items 1 to 10 of 25 total — Page: 1 2 3 ▸ — Show 10 per page

In order to create a Magento grid, you can try to get the data and display it in a common PHP table. However, it takes you a lot of time and effort to configure the template as well as pagination. This lesson will briefly introduce how to create a standard grid with available support from Magento.

For example, after installing Magento and creating a new module named Lesson13, we build a database table (table lesson13) with the fields: person_id, name, birthday, gender, and address with the following information (in phpMyAdmin)



←T→			person_id	name	birthday	gender	address
☐	✎	✗	1	Johnny Giap	1990-06-19	1	Viet Nam
☐	✎	✗	2	Michael Nguyen	1987-01-03	1	USA
☐	✎	✗	3	Blanka Pham	1990-08-19	1	Viet Nam
☐	✎	✗	4	Bruce Pham	1990-06-22	1	Viet Nam
☐	✎	✗	5	Luna Hoang	1987-03-29	0	England
☐	✎	✗	6	Sarah Nguyen	1990-01-14	0	Viet Nam
☐	✎	✗	7	Hannah Nguyen	1990-01-30	0	German
☐	✎	✗	8	David Nguyen	1987-04-16	1	Viet Nam
☐	✎	✗	9	Justin Pham	1990-07-25	1	Viet Nam
☐	✎	✗	10	Alex Nguyen	1997-03-18	1	India
☐	✎	✗	11	Travis Ngo	1984-06-06	1	Viet Nam
☐	✎	✗	12	Jenny Ta	1987-04-23	0	Laos

Model corresponding to this table is Lesson13/Model/Lesson13.php

Note: Methods to configure Model and database were introduced in Lesson 5.

2. Process to create a frontend grid

As mentioned in Lesson12, it takes four steps to build a template in frontend:

Step 1: Create a layout for action index: *app/design/frontend/default/default/lesson13.xml*

```
<layout version="0.1.0">
    <default>
        <!-- update layout for all frontend page -->
    </default>
    <lesson13_index_index>
        <reference name="content">
            <block type="lesson13/lesson13" name="lesson13"
template="lesson13/lesson13.phtml" />
        </reference>
    </lesson13_index_index>
</layout>
```

Step 2: Create controller/Action: indexAction() in file

app/code/local/Magestore/Lesson13/controllers/IndexController.php

```
class Magestore_Lesson13_IndexController extends
Mage_Core_Controller_Front_Action
{
    /**
     * index action
     */
    public function indexAction()
    {
        $this->loadLayout();
        $this->renderLayout();
    }
}
```

Step 3: Create Block Lesson 13 in file

app/code/local/Magestore/Lesson13/Block/Lesson13.php

```
class Magestore_Lesson13_Block_Lesson13 extends Mage_Core_Block_Template
{
    /**
     * prepare block's layout
     *
     * @return Magestore_Lesson13_Block_Lesson13
     */

    public function __construct()
    {
        parent::__construct();
        $collection = Mage::getModel('lesson13/lesson13')->getCollection();
        $this->setCollection($collection);
    }
}
```

Step 4: Create file template

app/design/frontend/default/default/template/lesson13/lesson13.phtml

File template will display all HTML paragraphs we want to show out in frontend.

3. Writing templates for grid

In this section, we will go into details: How the rows and columns are shown in the grid as described above.

```
<!-- Page Title -->
<div class="page-title">
    <h2><?php echo $this->__('Lesson13 Module Grid Template') ?></h2>
</div>

<table id="my-person-table" class="data-table">
    <colgroup>
        <col width="1">
        <col>
        <col width="1">
        <col width="1">
        <col>
    </colgroup>
    <thead>
```

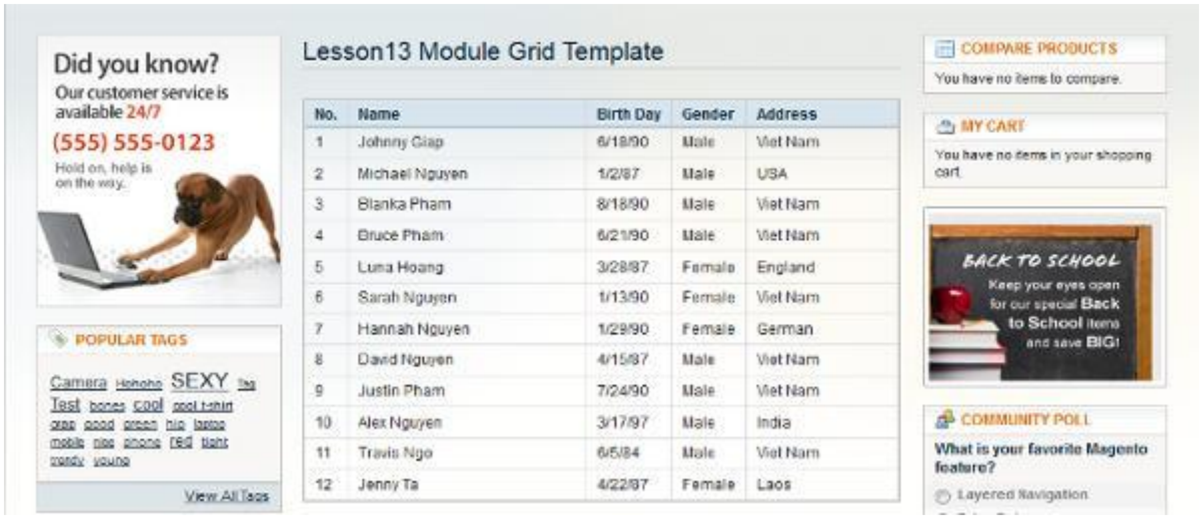
```
        <tr class="first last">
            <th><?php echo $this->__('No.')?></th>
            <th><?php echo $this->__('Name')?></th>
            <th><?php echo $this->__('Birth Day')?></th>
            <th><?php echo $this->__('Gender')?></th>
            <th><?php echo $this->__('Address')?></th>
        </tr>
    </thead>
    <tbody>
        <?php foreach($this->getCollection() as $person): ?>
            <tr>
                <td><?php echo $person->getData('person_id')?></td>
                <td><?php echo $person->getData('name')?></td>
                <td><?php echo Mage::helper('core')->formatDate($person-
>getData('birthday'))?></td>
                <td><?php echo $person->getData('gender')?></td>
                <td><?php echo $person->getData('address')?></td>
            </tr>
        <?php endforeach ?>
    </tbody>
</table>
```

An information table will be displayed in the code above. In this code, we should pay attention to this function:

```
Mage::helper('core')->formatDate($person->getData('birthday')).
```

This function can convert form of date in the database according to "currently locales" of store in config setting. In addition to formatDate function, there are a lot of other functions for coverting data into standard format which are written in: *app/code/core/Mage/Core /Helper/Data.php*

The result is as follow:



Did you know?
Our customer service is available 24/7
(555) 555-0123
Hold on, help is on the way.

Lesson13 Module Grid Template

No.	Name	Birth Day	Gender	Address
1	Johnny Giap	6/18/90	Male	Viet Nam
2	Michael Nguyen	1/2/87	Male	USA
3	Blanka Pham	8/18/90	Male	Viet Nam
4	Bruce Pham	6/21/90	Male	Viet Nam
5	Luna Hoang	3/28/87	Female	England
6	Sarah Nguyen	1/13/90	Female	Viet Nam
7	Hannah Nguyen	1/29/90	Female	German
8	David Nguyen	4/15/87	Male	Viet Nam
9	Justin Pham	7/24/90	Male	Viet Nam
10	Alex Nguyen	3/17/97	Male	India
11	Travis Ngo	6/5/84	Male	Viet Nam
12	Jenny Ta	4/22/87	Female	Laos

COMPARE PRODUCTS
You have no items to compare.

MY CART
You have no items in your shopping cart.

COMMUNITY POLL
What is your favorite Magento feature?
Layered Navigation

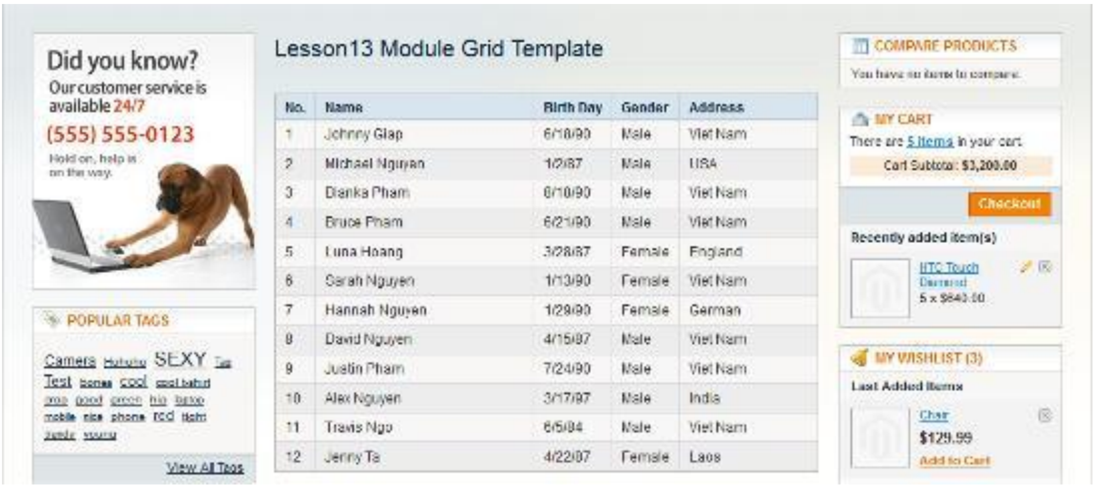
POPULAR TAGS
Camera Hohoho SEXY Tag
Test bones cool cool t-shirt
gift good green hip laptop
mobile nice phone red tight
trendy young
View All Tags
BACK TO SCHOOL
Keep your eyes open for our special Back to School items and save BIG!

Currently, rows in the above table are very difficult to distinguish from each other. That is why Magento is available to support a JavaScript function to differentiate the rows color. Here is the code:

```
<<script type="text/javascript">decorateTable('my-person-table')</script>
```

In which `'my-person-table'` is ID of the table.

The result will be:



Lesson13 Module Grid Template

No.	Name	Birth Day	Gender	Address
1	Johnny Giap	6/18/90	Male	Viet Nam
2	Michael Nguyen	1/2/87	Male	USA
3	Dianka Pham	6/10/90	Male	Viet Nam
4	Bruce Pham	6/21/90	Male	Viet Nam
5	Luna Hoang	3/28/87	Female	England
6	Sarah Nguyen	1/13/90	Female	Viet Nam
7	Hannah Nguyen	1/29/90	Female	German
8	David Nguyen	4/15/87	Male	Viet Nam
9	Justin Pham	7/24/90	Male	Viet Nam
10	Alex Nguyen	3/17/87	Male	India
11	Travis Ngo	6/5/84	Male	Viet Nam
12	Jenny Ta	4/22/87	Female	Laos

4. Grid pagination

For the tables with large amount of data/rows, we need to paginate for grid. In Magento, there is no need for great effort to calculate the number of rows, pages with normal PHP functions. We just need to go through some basic steps:

- Insert */app/code/core/Mage/Page/Block/Html/Pager.php* to create a sub-block of block lesson 13 which is already configured, and then name it (for example: lesson13_pager)

```
<layout version="0.1.0">
    <default>
        <!-- update layout for all frontend page -->
    </default>
    <lesson13_index_index>
        <reference name="content">
            <block type="lesson13/lesson13" name="lesson13"
template="lesson13/lesson13.phtml" >
                <block type="page/html_pager" name="lesson13_page"/>
            </block>
        </reference>
    </lesson13_index_index>
</layout>
```

- Rewrite two functions in block: prepareLayout() and getPagerHtml()

  app/code/local/Magestore/Lesson13/Block/Lesson13.php

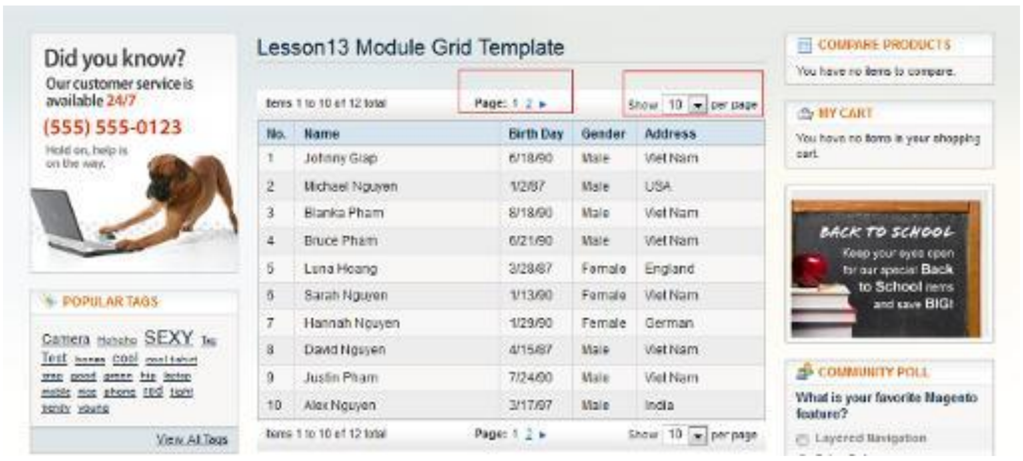
```
public function _prepareLayout()
    {
        parent::_prepareLayout();

        $pager = $this->getLayout()->createBlock('page/html_pager',
'lesson13.pager')->setCollection($this->getCollection());
        $this->setChild('pager', $pager);
        return $this;
    }

    public function getPagerHtml()
    {
        return $this->getChildHtml('pager');
    }
```

- Add paging to template file

  app/design/frontend/default/default/template/lesson13/lesson13.phtml

```
<!-- Page Title -->
<div class="page-title">
    <h2><?php echo $this->__('Lesson13 Module Grid Template') ?></h2>
</div>
<?php echo $this->getPagerHtml() ?>  <!--Add Paging-->
<table id="my-person-table" class="data-table">
    ……
</table>
<?php echo $this->getPagerHtml() ?> <!--Add Paging-->
<script type="text/javascript">decorateTable('my-person-table')</script>
```

The result is as follow:


Did you know?
Our customer service is available 24/7
(555) 555-0123
Hold on, help is on the way.
POPULAR TAGS
View All Tags
COMPARE PRODUCTS
MY CART
BACK TO SCHOOL
Keep your eyes open for our special Back to School items and save BIG!
COMMUNITY POLL
Layered Navigation

Lesson13 Module Grid Template

Items 1 to 10 of 12 total   Page: 1 2 ▸   Show 10 per page

No.	Name	Birth Day	Gender	Address
1	Johnny Giap	6/18/90	Male	Viet Nam
2	Michael Nguyen	1/2/87	Male	USA
3	Bianka Pham	8/18/90	Male	Viet Nam
4	Bruce Pham	6/21/90	Male	Viet Nam
5	Luna Hoang	3/28/87	Female	England
6	Sarah Nguyen	1/13/90	Female	Viet Nam
7	Hannah Nguyen	1/29/90	Female	German
8	David Nguyen	4/15/87	Male	Viet Nam
9	Justin Pham	7/24/90	Male	Viet Nam
10	Alex Nguyen	3/17/87	Male	India

Items 1 to 10 of 12 total   Page: 1 2 ▸   Show 10 per page

You have no items to compare.

You have no items in your shopping cart.

What is your favorite Magento feature?

III. SUMMARY AND REFERENCES

So, after learning this lesson, we know:

- Grid in Magento
- How to create a simple Grid
- Grid pagination

You are recommended to read more about this topic in:

- Lesson 11: Magento Configuration

- Lesson 12: Layout and Template on Magento Frontend

Back to top

Lesson 14: Magento Form on Frontend

I. INTRODUCTION

1. Time: 3 hours

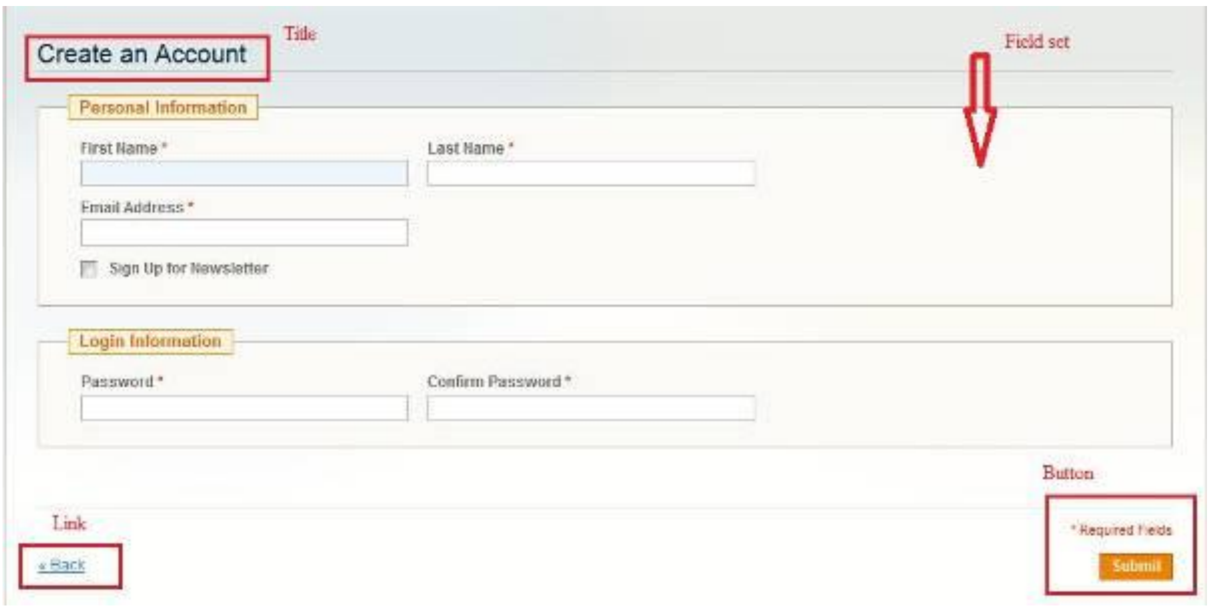
2. Content

- Process of creating a Magento form on frontend
- HTML structure of a normal Magento form
- How to name a class and create validate fields for a Magento form
- Data processing after form creation

II. CONTENT

1. Process to create a Magento form

An example of Account registration form:



A form with Magento standard often consists of three main parts:

- Form title
- Field set with blocks of information inside
- Button and Link

In lesson 13, you are already familiar with how to build a grid on Magento frontend. However, data extracted to grid is data queried from database (SQL). That is why I will guide you how to create a form to store that information in a database. Specifically, we will work with module *Lesson 14*.

Firstly, we build a database *Lesson 14* with these fields:

- person_id (Key): int
- name: varchar(255)
- birthday: date
- gender: varchar(255)
- address: varchar(255)

We create models to process data from this database. The configuration of these models was introduced in Lesson 5. In this lesson, Model working with Lesson 14 database is *app/code/local/Magestore/Lesson14/Model/Lesson14.php.*

Secondly, we configure controller/Action in layout file:

app/design/frontend/default/default/layout/lesson14.xml

```
<layout version="0.1.0">
    <lesson14_index_index>
        <reference name="root">
            <action method="setTemplate"><template>page/2columns-
left.phtml</template></action>
        </reference>
        <reference name="content">
            <block type="lesson14/lesson14" name="lesson14"
template="lesson14/lesson14.phtml" />
        </reference>
    </lesson14_index_index>
</layout>
```

Thirdly, create a file block: *app/code/local/Magestore/Lesson14/Block/Lesson14/php*

```
class Magestore_Lesson14_Block_Lesson14 extends Mage_Core_Block_Template
{
    public function getActionOfForm()
    {
        return $this->getUrl('lesson14/index/createPerson);
    }
}
```

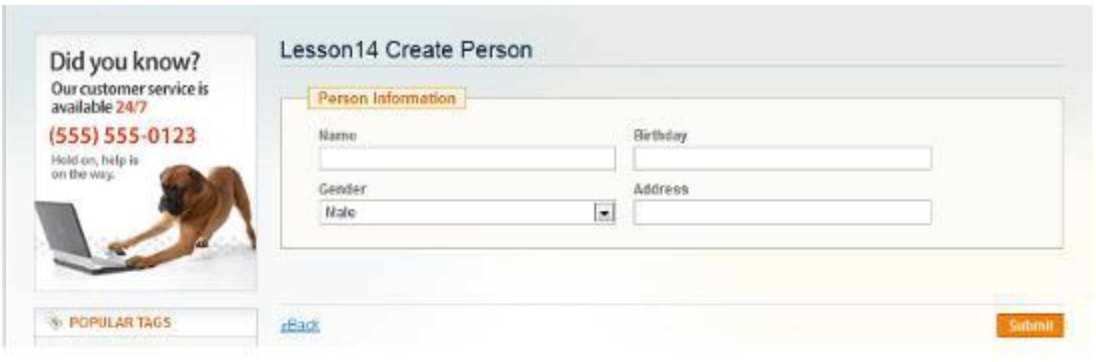
Finally, create a file template: *app/design/frontend/default/default/template/lesson14.phtml*

```
<!--Form Tittle-->
<div class="page-title">
    <h2><?php echo $this->__('Lesson14 Create Person') ?></h2>
```

```
</div>
<!--Form Information-->
<form id="person-form-validate" method="post" action="<?php echo $this-
>getActionOfForm()?>">
    <div class="fieldset">
        <h2 class="legend"><?php echo $this->__('Person
Information')?></h2>
        <ul class="form-list">
            <li class="fields">
                <div class="customer-name">
                    <div class="field name-firstname">
                        <label class="required"
for="firstname"><em>*</em><?php echo $this->__('Name')?></label>
                        <div class="input-box">
                            <input type="text" class="input-text required-
entry" maxlength="255" title="Name" value="" name="name" id="name">
                        </div>
                    </div>
                    <div class="field birthday">
                        <label class="required"
for="birthday"><em>*</em><?php echo $this->__('Birthday')?></label>
                        <div class="input-box">
                            <input type="text" class="input-text required-
entry" maxlength="255" title="Birthday" value="" name="birthday"
id="birthday">
                        </div>
                    </div>
                </div>
            </li>
            <li class="fields">
                <div class="gender">
                    <div class="field gender">
                        <label class="required"
for="gender"><em>*</em><?php echo $this->__('Gender')?></label>
                        <div class="input-box">
                            <input type="text" class="input-text required-
entry" maxlength="255" title="Gender" value="" name="gender" id="gender">
                        </div>
                    </div>
                    <div class="field address">
                        <label class="required"
for="address"><em>*</em><?php echo $this->__('Address')?></label>
                        <div class="input-box">
                            <input type="text" class="input-text required-
entry" maxlength="255" title="Address" value="" name="address"
id="address">
                        </div>
                    </div>
                </div>
            </li>
        </ul>
    </div>
    <!--Button & Link-->
    <div class="buttons-set">
        <p class="required">* <?php echo $this->__('Required Fields')?></p>
        <p class="back-link"><a class="back-link" href="<?php echo $this-
>getBaseUrl();?>"><small>&laquo;</small>Back</a></p>
        <button class="button" title="Submit"
type="submit"><span><span><?php echo $this-
>__('Submit')?></span></span></button>
    </div>
```

```
</form>
```

The result is as follow:



2. HTML structure of a normal Magento form

A Magento form usually contains default classes (in layout). The above example has shown a form with Magento standard:

- The outside is the form tag with method and action

- The inside of the form often has *fieldset* to classify groups with different information (such as in the registration form, there are two fieldsets: personal information and login information).

- Class *legend* is used to name Fieldset.

- Next is the *form-list* classes, and smaller is *fields* class containing the fields in the fieldset.

- Fields often contains labels and input HTML tags.

3. How to name class and create validate fields for a Magento form

In addition to help process interface support, Magento has many JavaScript functions that form required fields in forms, email fields, checkbox button, radio button and so on.

To use them, we need to declare a form object with a JavaScript function written in template file that contains form. This function has structure as follows:

```
var dataForm = new VarienForm(form-id',true);
```

Form-id is the id of the form we want to declare. Apply this to the example: *app/design/frontend/default/default/template/lesson14.phtml*

```
<script type="text/javascript">
    //<![CDATA[
        var dataForm = new VarienForm('person-form-validate', true);
    //]]>
```

```
</script>
```

After the form is declared, all fields in the form are checked automatically by Magento available JavaScript functions. They are written in the file *js/varien/form.js.*

Fields in checked range must have valid class names. Below I will introduce you a list of class names:

Validate Class Name	Invalid messages
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

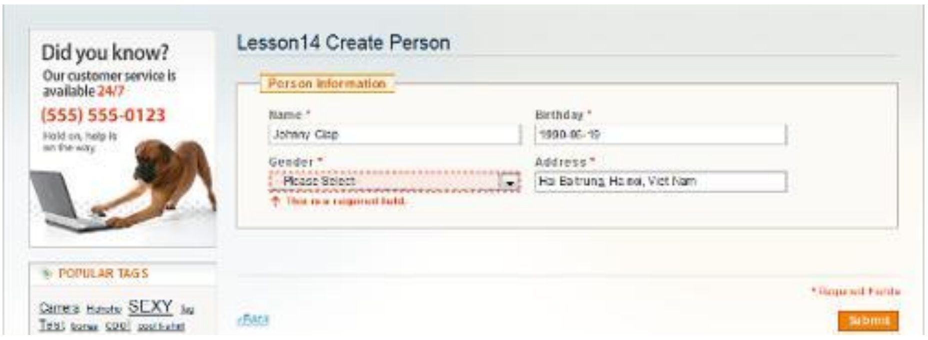
Validate Class Name	**Invalid messages**
validate-clean-url	Please enter a valid URL. For example http://www.example.com or www.example.com
validate-identifier	Please enter a valid Identifier. For example example-page, example-page.html or anotherlevel/example-page
validate-xml-identifier	Please enter 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.

In the example of module in lesson 14, we use two types of validate class: *required-entry and validate-date*. Below is a complete code block in the template file:

app/design/frontend/default/default/template/lesson14.phtml

```
<!--Form Tittle-->
<div class="page-title">
    <h2><?php echo $this->__('Lesson14 Create Person') ?></h2>
</div>
<!--Form Information-->
<form id="person-form-validate" method="post" action="<?php echo $this-
>getActionOfForm()?>">
    <div class="fieldset">
        <h2 class="legend"><?php echo $this->__('Person
Information')?></h2>
        <ul class="form-list">
            <li class="fields">
                <div class="customer-name">
                    <div class="field name-firstname">
                        <label class="required"
for="firstname"><em>*</em><?php echo $this->__('Name')?></label>
                        <div class="input-box">
                            <input type="text" class="input-text required-
entry" maxlength="255" title="Name" value="" name="name" id="name">
                        </div>
                    </div>
                    <div class="field birthday">
                        <label class="required"
for="birthday"><em>*</em><?php echo $this->__('Birthday')?></label>
                        <div class="input-box">
                            <input type="text" class="input-text required-
entry validate-date" maxlength="255" title="Birthday" value=""
name="birthday" id="birthday">
                        </div>
                    </div>
                </div>
            </li>
            <li class="fields">
                <div class="gender">
                    <div class="field gender">
                        <label class="required"
for="gender"><em>*</em><?php echo $this->__('Gender')?></label>
                        <div class="input-box">
                            <select type="text" class="input-text required-
entry" title="Gender" name="gender" id="gender">
                                <option value=""><?php echo $this->__('--
Please Select--')?></option>
                                <option value="Male"><?php echo $this-
>__('Male')?></option>
                                <option value="Female"><?php echo $this-
>__('Female')?></option>
                            </select>
                        </div>
                    </div>
                    <div class="field address">
                        <label class="required"
for="address"><em>*</em><?php echo $this->__('Address')?></label>
                        <div class="input-box">
                            <input type="text" class="input-text required-
entry" maxlength="255" title="Address" value="" name="address"
id="address">
                        </div>
                    </div>
                </div>
            </li>
        </ul>
    </div>
    <!--Button & Link-->
    <div class="buttons-set">
        <p class="required">* <?php echo $this->__('Required Fields')?></p>
        <p class="back-link"><a class="back-link" href="<?php echo $this->getBaseUrl();?>"><small>&laquo;</small>Back</a></p>
        <button class="button" title="Submit" type="submit"><span><span><?php echo $this->__('Submit')?></span></span></button>
    </div>
</form>
 <script type="text/javascript">
    //<![CDATA[
        var dataForm = new VarienForm('person-form-validate', true);
    //]]>
</script>
```

This is the result shown after you click on the *Submit* button:

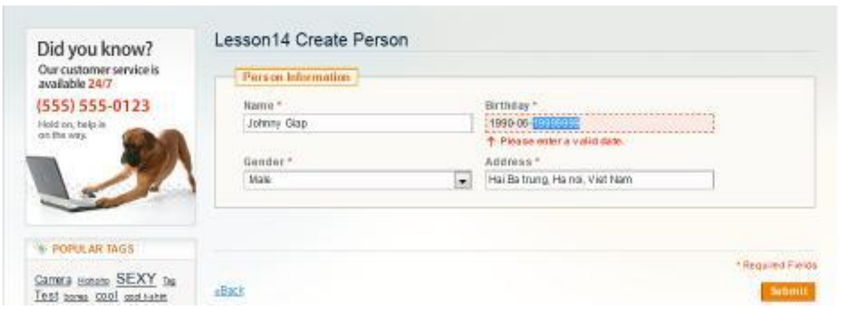

Lesson14 Create Person

Person Information

Name * : Johnny Giap
Birthday * : 1990-06-19
Gender * : Please Select
This is a required field.
Address * : Hai Ba trung, Ha noi, Viet Nam

* Required Fields

When required fields are not completed

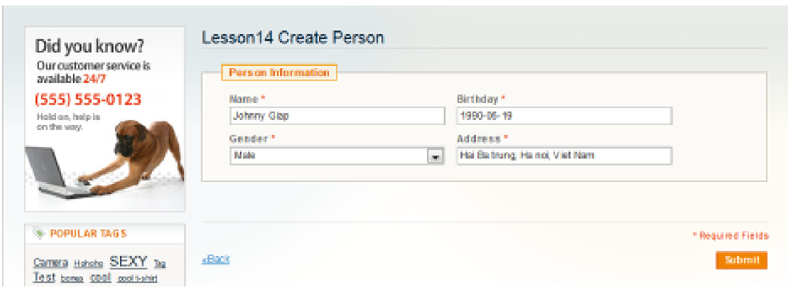

Lesson14 Create Person

Person Information

Name * : Johnny Giap
Birthday * : 1990-06-19999999
Please enter a valid date.
Gender * : Male
Address * : Hai Ba trung, Ha noi, Viet Nam

* Required Fields

When date format is not correct


Lesson14 Create Person

Person Information

Name * : Johnny Giap
Birthday * : 1990-06-19
Gender * : Male
Address * : Hai Ba trung, Ha noi, Viet Nam

* Required Fields

When all fields are full and correct

4. How to process data after creating form

In this case, we want to save information into table *lesson 14* via the form. To do that, we create a controller to process the Magento form. It is noted that you should name input-boxes similar to fields in database you want to save so that you can edit database with convenience later.

app/code/local/Magestore/Lesson14/controllers/IndexController.php

```
    public function createPersonAction()
    {
        $data = $this->getRequest()->getPost();
        $session = Mage::getSingleton('core/session');
        $person = Mage::getModel('lesson14/lesson14');
        $person->setData('name', $data['name']);
        $person->setData('birthday', $data['birthday']);
        $person->setData('gender', $data['gender']);
        $person->setData('address', $data['address']);
        // $person->setData($data);
        try{
            $person->save();
            $session->addSuccess('Add a person sucessfully');
        }catch(Exception $e){
            $session->addError('Add Error');
        }
        $this->_redirect('lesson14/index/index');
    }
```

III. SUMMARY AND REFERENCES

After participating in two parts of lesson 14, you should know:

- Overview of Magento form on frontend
- How to create a basic Magento form outside the frontend
- How to use validate field in Magento form
- How to handle data after submitting form

You can read the following materials to understand this lesson better:

- Lesson 11: Magento Configuration
- Lesson 12: Layout & Template on Frontend of Magento
- Magentoexpertforum.com, 2013. *Form validation in Magento*. [Online] Available at: http://magentoexpertforum.com/showthread.php/10076-Form-validation-in-Magento.

Back to top

Lesson 15: CSS and JavaScript in Frontend

I. INTRODUCTION

1. Time: 2.5 hours

2. Content:

- How to add a CSS file to frontend
- How to add a Javascript file to Frontend (Javascript in Javascript folder and Javascript in skin)
- Javascript and CSS on template

II. CONTENT

1. How to add a Magento CSS file to frontend

1.1. Creating a CSS file in folder skin/frontend/default/default/css/ or skin/frontend/base/default/css/

Example: skin/frontend/default/default/css/magestore/mycss.css

Or: skin/frontend/default/default/css/mycustom.css

Or: skin/frontend/base/default/css/mycustom.css.

1.2. Adding a CSS file from layout file

In the following examples, system firstly finds the declared CSS file in folder *skin/frontend/default/default/css*. If it cannot find in this folder, system automatically finds in *folder skin/frontend/base/default/css*. If the system fails to find in both above folders, it will report error or skip.

* For comprehensive system (always add when layout files have not been loaded)

```
<layout version="0.1.0">
    <default>
    <reference name="head">
                <action method="addCss">
                    <stylesheet>css/mycustom.css </stylesheet>
                    <params>media="all"</params>
                </action>
            <action method="addCss">
                    <stylesheet>css/magestore/mycss.css
                </stylesheet>
                </action>
```

```
         </reference>
     </default>
</layout>
```

The following code block will add the following files respectively:

- ✓ skin/frontend/base/default/css/mycustom.css
- ✓ skin/frontend/base/default/css/magestore/mycss.css to <head></head> when this module is loaded.

* For each site (only add when this action/page is loaded)

```
<layout version="0.1.0">
    <lesson15_index_index>
        <reference name="head">
            <action method="addCss">
                    <stylesheet>css/mycustom.css </stylesheet>
                    <params>media="all"</params>
            </action>
    </lesson15_index_index>
</layout>
```

The above code block will add file mycustom.css to index action, controller index of module lesson 15.

Besides method='addCSS', we can use other methods to add CSS from outside CSS library as follows:

```
<?xml version="1.0"?>
    <layout version="0.1.0">
          <default>
               <reference name="head">
                     <block type="core/text" name="addjquery">
                           <action method="setText">
                                 <text>
<![CDATA[<link href="http://yourhostcss/js/calendar/calendar-
win2k-1.css" type="text/css" rel="stylesheet"> ]]>
                                 </text>
                           </action>
                     </block>
               </reference>
         </default>
</layout>
```

2. How to add a Javascript file to frontend

2.1 Adding a Javascript file in Javascript folder

```
   <?xml version="1.0"?>
       <layout version="0.1.0">
               <lesson15_index_test>
                   <reference name="head">
                           <action method="addJs">
                                 <script>yourscript.js</script>
                           </action>
                       </reference>
               </lesson15_index_test >
```

```
                <lesson15_index_view>
                        <reference name="head">
                    <action method="addJs">
                        <script>yourscript2.js</script>
                    </action>
                        </reference>
                </lesson15_index_view >
        </layout>
```

The above code block will add JavaScript in file *js/yourscript.js* to <head></head> when action test of module lesson 15 is loaded, and add JavaScript in file *js/yourscript2.js* to <head></head> when action view of module lesson 15 is loaded.

2.2. Adding a Javascript file in skin folder

```
    <?xml version="1.0"?>
        <layout version="0.1.0">
        <lesson15_index_test>
             <reference name="head">
                <action  method="addItem">
                    <type>skin_js</type>
                    <name>path/newfile.js</name>
                </action>
            </reference>
        </lesson15_index_test >
    </layout>
```

The above code block will add JavaScript in file *skin/frontend/base/default/path/newfile.js* to <head></head> when action test of module lesson 15 is loaded.

2.3. Adding a JavaScript file from an outside library

```
<?xml version="1.0"?>
    <layout version="0.1.0">
            <default>
            <reference name="head">
                <block type="core/text" name="addjquery">
                    <action method="setText">
                        <text>
<![CDATA[<script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js
"></script>
<script type="text/javascript">jQuery.noConflict();</script>]]>
                    </text>
                </action>
            </block>
        </reference>
</default>
</layout>
```

3. How to add a Magento CSS file and a Magento Javascript file from file controller

3.1. Adding a Magento CSS file from file controller

```
public function indexAction() {
    $this->loadLayout();
    $head = Mage::app()->getLayout()->getBlock('head');
    $head->addItem('skin_css', 'css/additional.css');

    $this->renderLayout();
}
```

The above code block will add CSS in file kin/frontend/base/default/css/additional.css to <head></head> when action index of module lesson 15 is loaded.

3.2. Adding a Magento Javascript file from file controller

```
public function indexAction() {
    $this->loadLayout();

    $head = Mage::app()->getLayout()->getBlock('head');
    $head->addItem('skin_js', 'js/additional.js');

    $this->renderLayout();
}
```

The above code block will add JS in file *skin/frontend/base/default/js/additional.js* to <head></head> when action index of module lesson 15 is loaded.

4. How to add a Magento CSS file and Magento JavaScript file to a page from block

4.1. Adding a Magento CSS file from block

```
class Magestore_Lesson15_Block_Lesson15 extends
Mage_Core_Block_Template
{
    public function addCss(){
         $this->getLayout()->getBlock('head')
->addCss('path/from/css/folder/to/your/file.css');
    }
}
```

4.2. Adding Magento JavaScript file from block

```
class Magestore_Lesson15_Block_Lesson15 extends
Mage_Core_Block_Template
{
    public function addJs(){
         $this->getLayout()->getBlock('head')-
>addJs('js/myjsfile.js');
    }
}
```

5. How to add a JavaScript file and CSS file to a Magento page from template

5.1. Adding a CSS file directly on file template

```
<link href= "path/from/css/folder/to/your/file.css " rel="stylesheet"
type="text/css">
```

5.2. Adding a JavaScript file directly on file template

Add the following code block to template file:

```
<script type="text/javascript" src="<?php echo $this ->
getSkinUrl('js/functions.js') ?>"></script>
```

6. Some notes of Magento Javascript file

6.1. JSON

Encode from an array:

```
$jsonData = Mage::helper('core')->jsonEncode($array);
```

Decode to an array:

```
$array = Mage::helper('core')->jsonDecode($jsonData);
```

In your Magento Controller/Action, you can use the below code to send a JSON response:

```
$this->getResponse()->setHeader('Content-type', 'application/json');
$this->getResponse()->setBody($jsonData);
```

6.2. JsQuoteEscape

If you want to assign a string "He says: I'm Ron." to a JS variable, you need to use jsQuoteEscape() function.

```
<?php
$magestoreSlogan = "'magento extensions of innovative functions' is
Magestore's slogan";
?>
<script>
    var escapejs = '<?php echo Mage::helper('core')-
>jsQuoteEscape($magestoreSlogan)  ;?>';
    alert(escapejs);
</script>
```

III. SUMMARY

We will gain some knowledge after learning this lesson:

- Configuring layout to add a custom CSS in skin folder for the whole Magento system or each page
- Configuring layout to add CSS from an outside CSS library to Magento pages
- Configuring layout to add JavaScript to a page or the whole Magento system, JavaScript file is either in JavaScript folder or in skin folder.
- Adding CSS and JavaScript directly in template.

Back to top

Lesson 16: Magento Email Templates

I. INTRODUCTION

1. Time: 2 hours

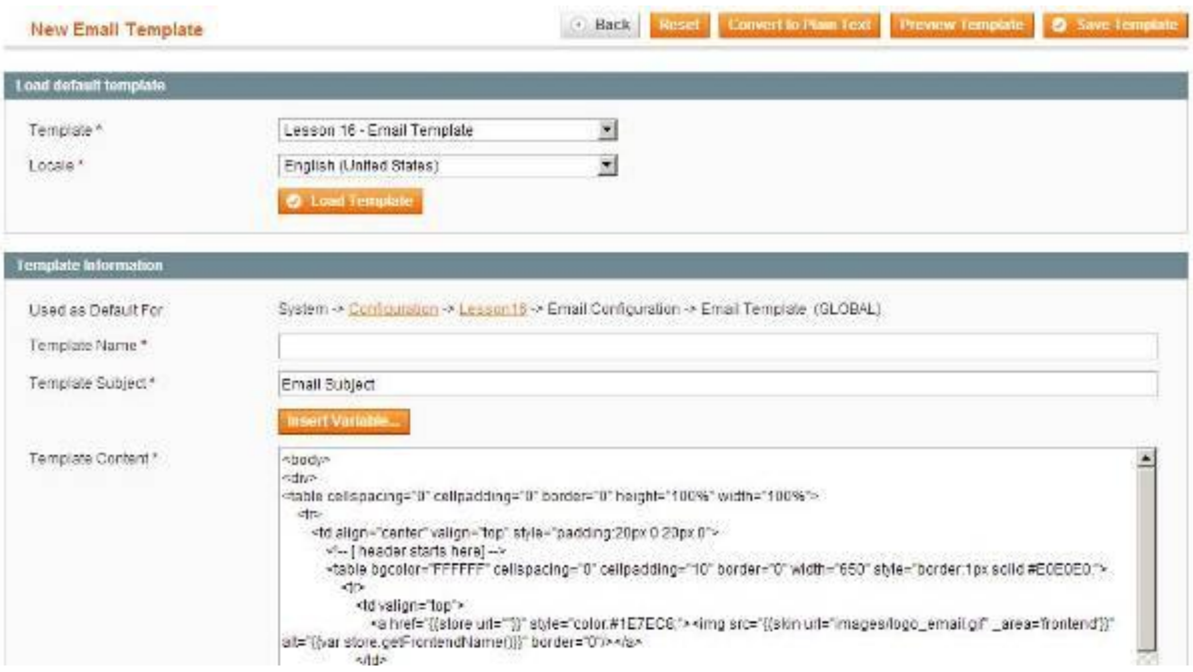
2. Content

- How to add Magento email templates
- How to write Magento email templates
- How to send an email
- Variables in Magento email templates.

II. CONTENT

1. How to add Magento email templates

When an administrator wants to create a new email template, he accesses **System/Transactional Emails** and chooses **Add New Template**. He can load the content of available templates to make new ones based on them.



Template *	Lesson 16 - Email Template
Locale *	English (United States)
Used as Default For	System -> Configuration -> Lesson16 -> Email Configuration -> Email Template (GLOBAL)
Template Name *	
Template Subject *	Email Subject

Template Content *

```
<body>
<div>
<table cellspacing="0" cellpadding="0" border="0" height="100%" width="100%">
  <tr>
    <td align="center" valign="top" style="padding:20px 0 20px 0">
      <!-- [ header starts here] -->
      <table bgcolor="FFFFFF" cellspacing="0" cellpadding="10" border="0" width="650" style="border:1px solid #E0E0E0;">
        <tr>
          <td valign="top">
            <a href="{{store url=""}}" style="color:#1E7EC8;"><img src="{{skin url="images/logo_email.gif" _area='frontend'}}"
alt="{{var store.getFrontendName()}}" border="0"/></a>
          </td>
```

To add an email template file, we use the configuration in file config.xml in folder *(app/code/local/Magestore/Lesson16/etc)* of module with xpath *config/global/template/email*:

```
<config>
    ...
    <global>
        ...
        <template>
            <email>
                <lesson16_email_template translate="label"
module="lesson16">
                    <label>Lesson 16 - Email Template</label>
                    <file>lesson16.html</file>
                    <type>html</type>
                </lesson16_email_template>
            </email>
        </template>
    </global>
</config>
```

- ✓ Lesson 16_email_template: Identity of the email template declared
- ✓ Label: Email template's name
- ✓ File: Email template file's name. Magento finds this file in folder *app\locale\en_US\template\email*
- ✓ Type: Type of template file

After this step, we create a template file named *lesson16.html* in folder *app\locale\en_US\template\email.*

Like other typical configurations in Magento configuration system, we add an email template configuration to file system.xml in folder etc of module:

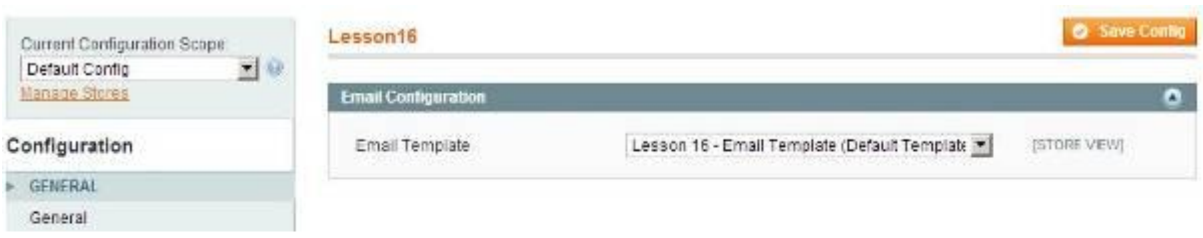
```
<config>
    <tabs>
        <magestore translate="label">
            <label>Magestore Extension</label>
            <sort_order>400</sort_order>
        </magestore>
    </tabs>
    <sections>
        <lesson16 translate="label" module="lesson16">
            <class>separator-top</class>
            <label>Lesson16</label>
            <tab>magestore</tab>
            <frontend_type>text</frontend_type>
            <sort_order>299</sort_order>
            <show_in_default>1</show_in_default>
            <show_in_website>1</show_in_website>
            <show_in_store>1</show_in_store>
            <groups>
                <email translate="label">
                    <label>Email Configuration</label>
                    <frontend_type>text</frontend_type>
                    <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>
                    <fields>
                        <template translate="label">
                            <label>Email Template</label>
                            <frontend_type>select</frontend_type>
                            <sort_order>10</sort_order>

<source_model>adminhtml/system_config_source_email_template</source_model>
                            <show_in_default>1</show_in_default>
                            <show_in_website>1</show_in_website>
                            <show_in_store>1</show_in_store>
                            <comment></comment>
                        </template>
                    </fields>
                </email>
            </groups>
        </lesson16>
    </sections>
</config>
```

The *source_model* used for email template is *adminhtml/system_config_source_email_template.*Source model receives value from template configuration (identity of template is *[session_code]_[group_code]_[template]*).

Finally, we need to add code to file *config.xml* and to folder etc of module in order to set a default value configuration for configuration field.

```
<config>
    ...
    <global>
        ...
    </global>
    <default>
        <lesson16>
            <email>
                <template>lesson16_email_template</template>
            </email>
        </lesson16>
    </default>
</config>
```

The default value is the identity of template declared. We receive an email template configuration in configuration system as follows:



Email Template: Lesson 16 - Email Template (Default Template [STORE VIEW]

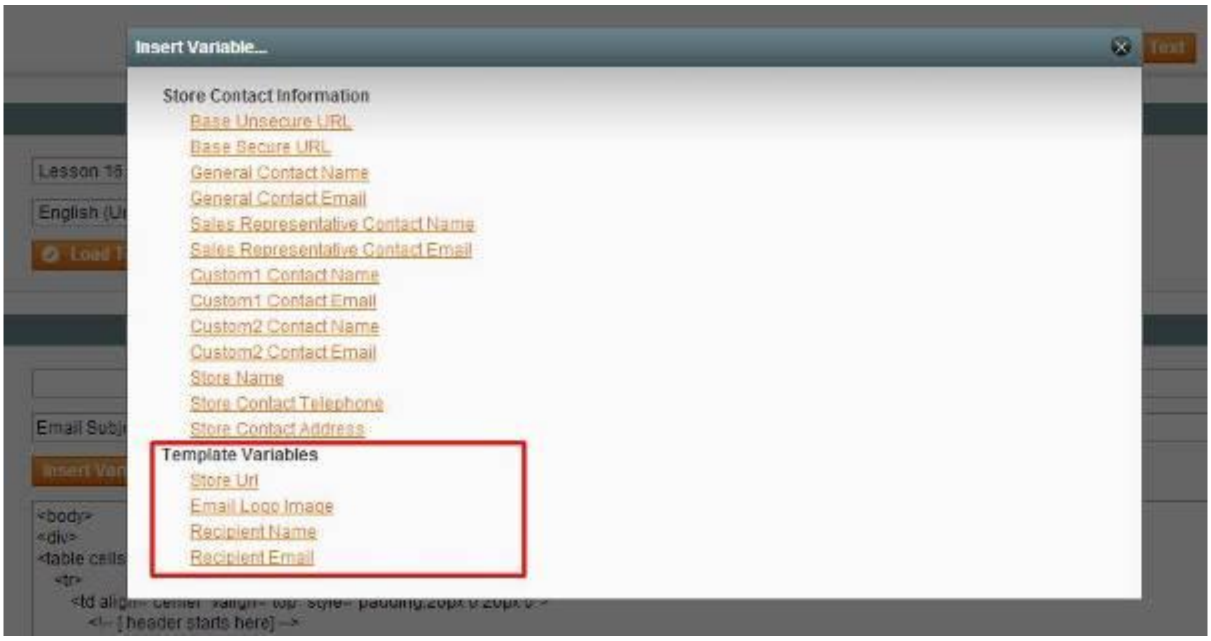
2. How to write Magento email templates

In the above example, we mentioned the email template file *lesson16.html* in *app\locale\en_US\template\email* folder. The structure of Magento email template file:

```
<!--@subject Email Subject @-->
<!--@vars
{"store url=\"\"":"Store Url",
"skin url=\"images/logo_email.gif\" _area='frontend'":"Email Logo Image",
"htmlescape var=$recipient_name":"Recipient Name",
"htmlescape var=$recipient_email":"Recipient Email"}
@-->
<!--@styles
body {background: #F6F6F6; margin: 0; padding: 0;}
body, td { color:#2f2f2f; font:11px/1.35em Verdana, Arial, Helvetica, sans-
serif; }
@-->

<body>
<div>
<table cellspacing="0" cellpadding="0" border="0" height="100%"
width="100%">
    <tr>
        <td align="center" valign="top" style="padding:20px 0 20px 0">
            <!-- [ header starts here] -->
            <table bgcolor="FFFFFF" cellspacing="0" cellpadding="10"
border="0" width="650" style="border:1px solid #E0E0E0;">
                <tr>
                    <td valign="top">
                        <a href="{{store url=""}}"
style="color:#1E7EC8;"><img src="{{skin url="images/logo_email.gif"
_area='frontend'}}" alt="{{var store.getFrontendName()}}" border="0"/></a>
                    </td>
                </tr>
                <!-- [ middle starts here] -->
                <tr>
                    <td valign="top">
                        <h1 style="font-size:22px; font-weight:normal;
line-height:22px; margin:0 0 11px 0;">Dear {{htmlescape
var=$recipient_name}},</h1>
                        This is the email that send from our magento
system.
                    </td>
                </tr>
                <tr>
                    <td bgcolor="#EAEAEA" align="center"
style="background:#EAEAEA; text-align:center;"><center><p style="font-
size:12px; margin:0;">Thank you again, <strong>{{var
store.getFrontendName()}}</strong></p></center></td>
                </tr>
            </table>
        </td>
    </tr>
</table>
</div>
</body>
```

✓ Subject is put in *<!–@subject and @–>* and is the Subject of an email when this template is used.

✓ Variables are put in *<!–@vars and @–>* and are values used when administrator makes a new template.



Store Contact Information
Base Unsecure URL
Base Secure URL
General Contact Name
General Contact Email
Sales Representative Contact Name
Sales Representative Contact Email
Custom1 Contact Name
Custom1 Contact Email
Custom2 Contact Name
Custom2 Contact Email
Store Name
Store Contact Telephone
Store Contact Address
Template Variables
Store Url
Email Logo Image
Recipient Name
Recipient Email

- ✓ Styles put in <*!–@styles* and @–> are CSS code of template.
- ✓ Content is the subject matter of an email. Typically, it begins with the code block <*body*> and ends with code block <*/body*>. In a content block containing dynamic content, its format is such as *{{htmlescape var=$recipient_name}}* or *{{var recipient_name}}*.

3. How to send an email

To write the function to send emails in Magento, we use the model 'core/ email_template'. In this model, we should notice two methods:

- setDesignConfig

```
public function setDesignConfig(array $config)
```

This method is used to initialize design information for template processing. The input parameter is a design configuration array *$config*.

- sendTransactional

```
public function sendTransactional($templateId, $sender, $email, $name,
$vars=array(), $storeId=null)
```

This method is the method to perform sending emails to recipients with parameters:

- ✓ *templateId*: the template identification
- ✓ *sender*: the sender's email address. This can be provided as a config part (to email configuration of Magento) or an array with the following form:

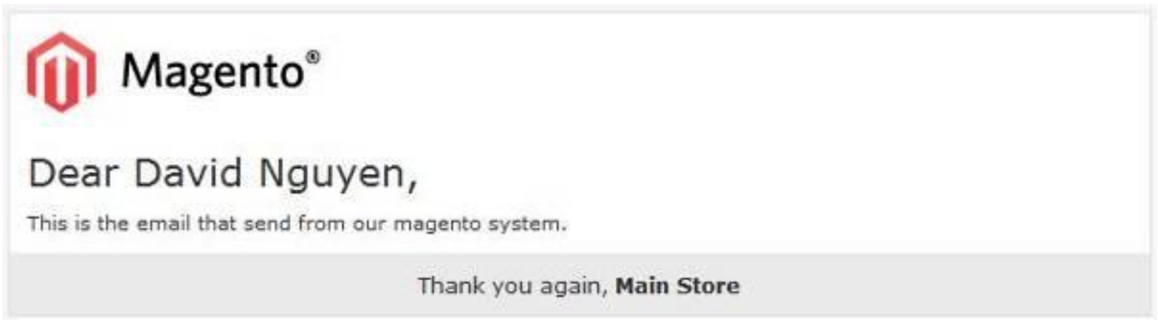
```
array(
    'name'   => 'Sender Name',
    'email'  => 'Sender Email (email@sample.com)'
)
```

- ✓ *email*: the recipient's email address
- ✓ *name*: the recipient's name
- ✓ *vars*: the values provided to the template when sending email. These values are correspondingly the variables we have used when writing the file template.
- ✓ *storeId*: the store that email is sent from. If not provided, this value is get from our design configuration (setDesignConfig)

Send an email in a controller:

```
class Magestore_Lesson16_IndexController extends
Mage_Core_Controller_Front_Action
{
    public function indexAction()
    {
        $recipientName  = 'David Nguyen';
        $recipientEmail = 'test@magestore.com';

        $store = Mage::app()->getStore();
        $translate = Mage::getSingleton('core/translate');
        $translate->setTranslateInline(false);

        Mage::getModel('core/email_template')
            ->setDesignConfig(array(
                'area'   => 'frontend',
                'store'  => $store->getId()
            ))->sendTransactional(
                Mage::getStoreConfig('lesson16/email/template', $store),
                Mage::getStoreConfig('trans_email/ident_general', $store),
                $recipientEmail,
                $recipientName,
                array(
                    'store'              => $store,
                    'recipient_name'     => $recipientName,
                    'recipient_email'    => $recipientEmail
                )
            );

        $translate->setTranslateInline(true);

        Mage::getSingleton('core/session')-
>addSuccess(Mage::helper('lesson16')->__('Email has been sent
successfully!'));

        $this->loadLayout();
        $this->renderLayout();
    }
}
```

Then we run the link on our web browser to this controller/action (for example http://localhost.com/magento1702/index.php/lesson16) and receive an email like this:


Dear David Nguyen,

This is the email that send from our magento system.

Thank you again, **Main Store**

4. Variables in Magento email template

In Magento email templates, Magento supports developers with programmable email content.

4.1 Putting content to email from Magento layout system

There are two ways to put content from design system into an email template: by block and by layout handle.

To put in by block, add this code to email template:

```
{{block type='lesson16/lesson16' area='frontend'
template='lesson16/lesson16.phtml' name=$recipient_name }}
```

- ✓ *type*: the block type
- ✓ *area*: the active area of the design (adminhtml, frontend)
- ✓ *template*: the path to the template file
- ✓ *name*: the properties provided to blog from template

To put in by layout, add this code to email template:

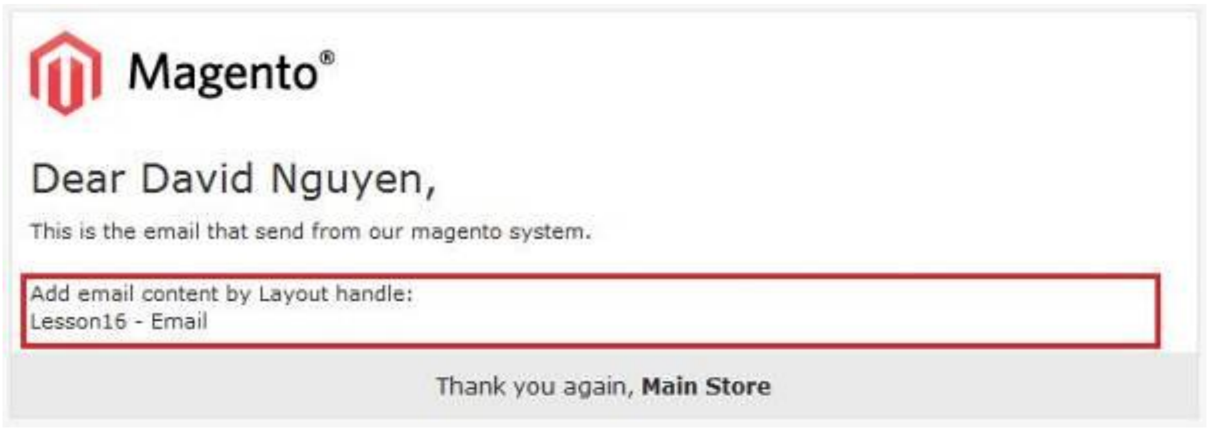
```
{{layout area="frontend" handle="lesson16_email"}}
```

- ✓ *area*: the active area of the design (adminhtml, frontend)
- ✓ *handle*: the name of the handle received from the layout system

In this code, we need to declare the handle *lesson16_email* from the file layout of the module *(app/design/frontend/default/default/layout/lesson16.xml)*:

```
<layout version="0.1.0">
    <lesson16_email><!-- handle used for email -->
        <reference name="content">
            <block type="lesson16/lesson16" name="lesson16"
template="lesson16/lesson16.phtml" />
        </reference>
    </lesson16_email>
</layout>
```

When an email is sent, the content of the blocks in this handle will be rendered, and the result will be:



Dear David Nguyen,

This is the email that send from our magento system.

Add email content by Layout handle:
Lesson16 - Email

Thank you again, **Main Store**

4.2 Adding URLs of stores and skins

We usually use the methods to get Base URL of stores or URL of files in the skin folder.

```
{{skin url="images/logo_email.gif" _area='frontend'}}
```

The above piece of code will send back an URL to *logo_email.gif* in the skin folder.

```
{{store url=""}}
```

And this piece of code will send an URL to the current store from which the email is sent.

4.3 Calling the method from the object

If the value provided when the email is sent is an object (in the above example, we provided in to 'store' as an object), we can also call the methods of that object like this:

```
{{var store.getFrontendName()}}
```

4.4 Depend Condition

There are two types of depend conditions in an email template supported by Magento: *depend* and *if* ... *else*. The syntax to use them is:

```
{{depend recipient_name}}
<!-- content -->
{{/depend}}
```

'Content' is showed only when 'recipient_name' is considered not to be empty.

```
{{if recipient_name}}
    <!-- content with recipient_name -->
{{else}}
    <!-- content without recipient_name -->
{{/if}}
```

In the above syntax, *'else'* is optional.

III. SUMMARY AND REFERENCES

In this lesson we have studied:

- How to create Magento email templates

- How to code to send email from Magento to a customer

To have a deep understanding of this lesson you can read:

- Lesson 11: Magento Configuration
- Magentocommerce, 2011. *Transactional emails*. [Online] Available at http://www.magentocommerce.com/wiki/modules_reference/english/mage_adminhtml/system_email_template/index.

Back to top

Lesson 17: Magento events

I. INTRODUCTION

1. Time: 2 hours

2. Content:

- An introduction to Magento events
- How to declare Magento observer (listener)
- Common Magento events

II. CONTENT

1. Magento Events

We can consider an event as a kind of flag that rises when a specific situation happens. For example, when a user presses the Place Order button on website, it is an event. Or your account has been created, which is also an event.

For example, Event *catalog_controller_category_init_before* is declared in file controller *app/code/core/Mage/Catalog/controllers/CategoryController.php* before initializing category object.

```
class Mage_Catalog_CategoryController extends
Mage_Core_Controller_Front_Action
{
    /**
     * Initialize requested category object
     *
     * @return Mage_Catalog_Model_Category
     */
    protected function _initCategory()
    {
        Mage::dispatchEvent('catalog_controller_category_init_before',
array('controller_action' => $this));
        ...
    }
}
```

In the above example, function Mage::dispatchEvent() raises a new event. It calls all observer callbacks registered for this event and multiple observers matching event name pattern.

Parameters in the Mage::dispatchEvent() function include:

- ✓ catalog_controller_category_init_before: Event name pattern
- ✓ array('controller_action' => $this): Data supplied for observers to listen to this event

2. Magento Observer

2.1. Definition

An observer is as a listener. It listens to your program to detect events which match name pattern. When an event takes place, all observers of that event are performed.

2.2. How to create a Magento observer

Declare a Magento observer

The code declaring observer is written in file *config.xml* of module. In this lesson, we need to create a *Magestore_Lesson17* module (you can use Module creator). Next, add the following code block to file app/code/local/Magestore/Lesson17/etc/config.xml.

```
<config>
   <frontend>
...

<events>
    <catalog_product_get_final_price>
        <observers>
            <magestore_lesson17_obsever>
                <type>singleton</type>
                <class>lesson17/observer</class>
                <method>getProductFinalPrice</method>
            </magestore_lesson17_observer>
        </observers>
    </catalog_product_get_final_price>
</events>

...
   </frontend>
</config>
```

Note: The above code block (in tags <events></events>) is added to tags <frontend></frontend>, which means that the observer declared will listen to events trigger in frontend. Moreover, we can write the code block used to declare the observer in tags <adminhtml></adminhtml> or <global></global>, corresponding to the scope of listened events (backend or both frontend and backend).

The parameters declared include:

- **catalog_product_get_final_price:** Name pattern to which the observer listens is the first parameter in function Mage:: dispatchEvent() when registering an event.
- **<type>singleton</type>:** It is the way to call the observer. It can be model, singleton or object
- **<class>lesson17/observer</class>:** This class is used to implement observer. In this example, class Magestore_Lesson17_Model_Observer declared in file app/code/local/Magestore/Lesson17/Model/Observer.php
- **<method>getProductFinalPrice</method>:** This method is called when event catalog_product_get_final_price takes place.

Write Observer Model

Create a file class app/code/local/Magestore/Lesson17/Model/Observer.php as follows:

```
<?php
class Magestore_Lesson17_Model_Obsever
{
    public function getProductFinalPrice($observer)
    {
        $event = $observer->getEvent();
        $product  = $event->getProduct();
        $product->setData('final_price',100);
    }
}
?>
```

In function *getProductFinalPrice()*, we can get data/objects provided by the event through parameter $observer ($observer is the object).

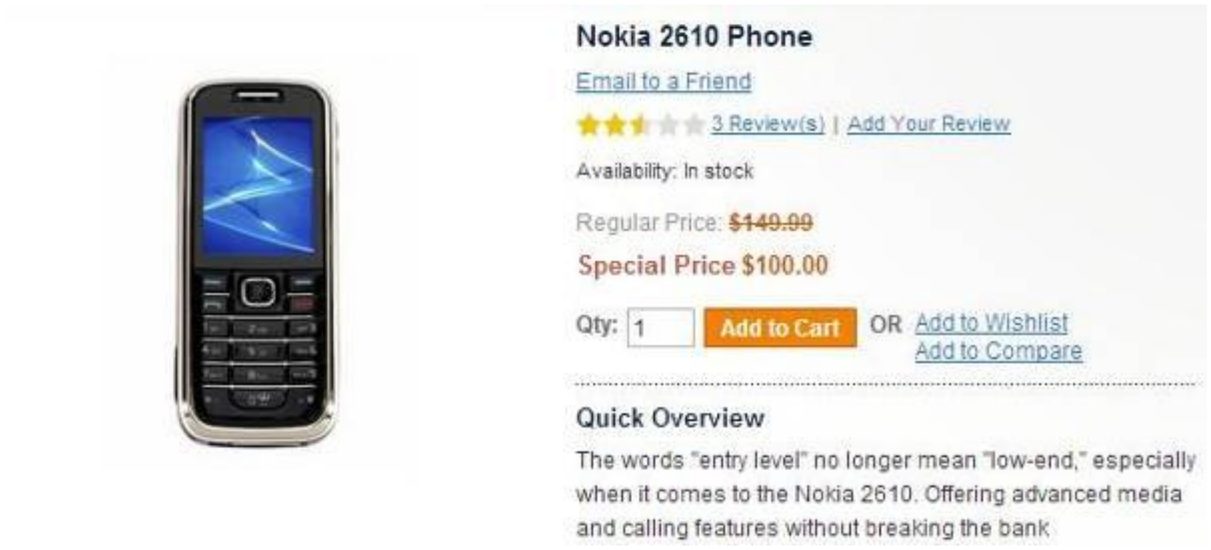
Look at the code block declared in event *catalog_product_get_final_price* to understand why we can get Order and Quote. Open file *app/code/core/Mage/Catalog/Model/Product/Type/Price.php*, then search for "catalog_product_get_final_price".

```
Mage::dispatchEvent('catalog_product_get_final_price', array('product' =>
$product, 'qty' => $qty));
```

The above command provides Product object and variable $qty for function Mage::dispatchEvent(). Thus, we can use them in this observers listen event.

Note: Product Object in function *getProductFinalPrice()* is a reference object. As a result, when we change attribute value of *$product* in this function, attribute value of product object transferred from function *Mage::dispatchEvent()* is changed, too.

Result shown on frontend product pages is as below. Final price of product is assigned $100.


Nokia 2610 Phone
Email to a Friend
3 Review(s) | Add Your Review
Availability: In stock
Regular Price: ~~$149.99~~
Special Price $100.00
Qty: 1
Add to Cart
OR Add to Wishlist
Add to Compare
Quick Overview
The words "entry level" no longer mean "low-end," especially when it comes to the Nokia 2610. Offering advanced media and calling features without breaking the bank

3. Common events in Magento

3.1. catalog_product_get_final_price

File path: *app/code/core/Mage/Catalog/Model/Product/Type/Price.php*

This event allows observers change final price of product.

```
Mage::dispatchEvent('catalog_product_get_final_price', array('product' =>
$product, 'qty' => $qty));
```

3.2. catalog_product_collection_apply_limitations_after

File path: *app/code/core/Mage/Catalog/Model/Resource/Product/Collection.php*

This event allows observers filter product collection before showing on frontend.

```
Mage::dispatchEvent('catalog_product_collection_apply_limitations_after',
array('collection'    => $this));
```

3.3. checkout_type_onepage_save_order_after

File path: *app/code/core/Mage/Checkout/Model/Type/Onepage.php*

This event allows observers to get data from Order, Quote or to change data of Order such as adding a discount total.

```
Mage::dispatchEvent('checkout_type_onepage_save_order_after',
                array('order'=>$order, 'quote'=>$this->getQuote()));
```

3.4. controller_action_predispatch

File path: *app/code/core/Mage/Core/Controller/Varien/Action.php*

This event always occurs when you access into any frontend page of Magento and is often used to carry out global website checkings.

```
Mage::dispatchEvent('controller_action_predispatch',
array('controller_action' => $this));
```

III. SUMMARY

This lesson brings us knowledge of

- Magento events
- How to declare Magento listener

Back to top

Lesson 18: Class Override in Magento

I. OVERVIEW

1. Time: 3 hours

2. Content:

- Overriding principle in Magento
- Overriding in Magento (Block, Model, ResourceModel, Helper)

II. CONTENT

1. Overriding principle in Magento

The overriding principle in Magento is very simple; it rewrites available functions in core to meet the user's purpose.

The rewrite of *Models/Resource models/Helper/Block* only applies to objects called through Mage, such as Mage::getModel(), Mage::getResourceModel(), Mage::helper(), Mage::getSingletonBlock().

This rewrite does not work with other ways of creating object such as new ($product = new Mage_Catalog_Model_Product();) or extends. This is the difference when we call for an object directly from its class name or using functions that Magento provides.

2. Overriding in Magento (Block, Model, Resource Model, Helper)

2.1. Overriding Block Class

Suppose that we want to offer a 20% discount on goods in the opening ceremony. To do that, we override class *Mage_Catalog_Block_Product_List* and show the special price of goods.

Firstly, declare that we override the above class by adding the following code block into tags <blocks><blocks> in file config.xml:

```
<config>
  <global>
      <blocks>
      <lesson18>
            <class>Magestore_Lesson18_Block</class>
      </lesson18>
      <catalog>
            <rewrite>
                  <product_list>
                  Magestore_Lesson18_Block_Catalog_Product_List
```

```
                   </product_list>
             </rewrite>
       </catalog>
       </blocks>
    </global>
</config>
```

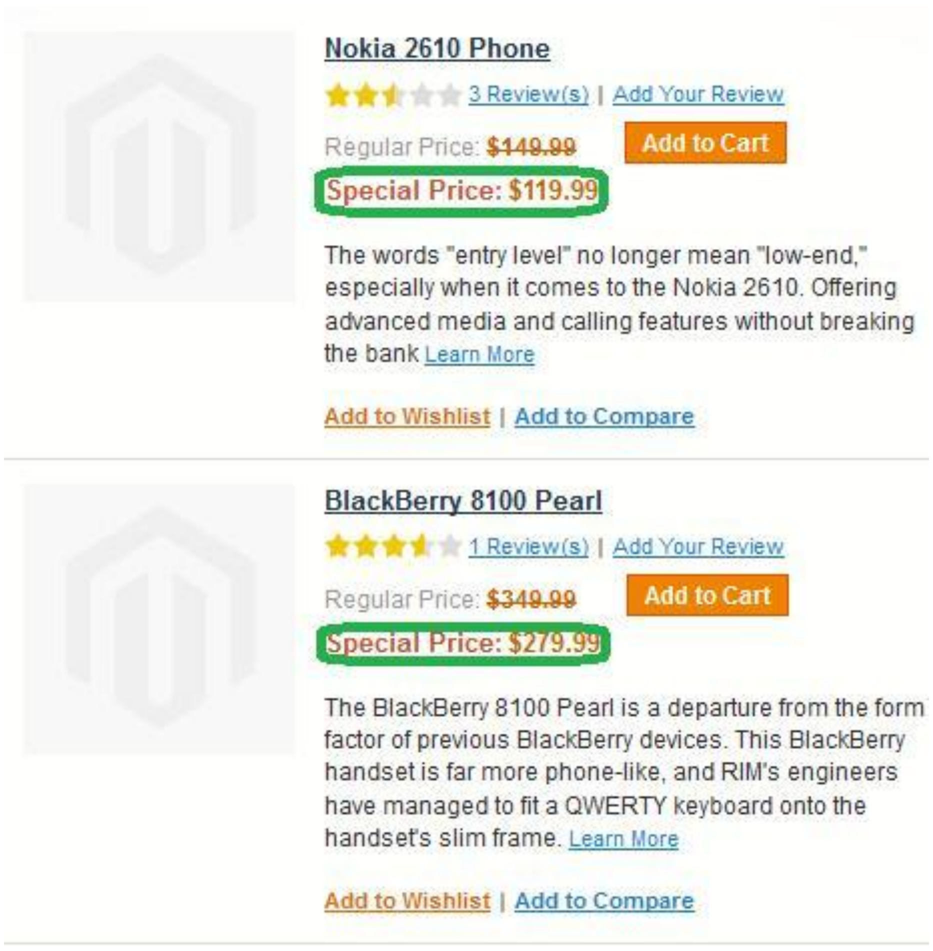
As you can see, we write “rewrite” tag inside “catalog” tag. “Catalog” tag implies app/code/core/Mage/Catalog/. Then, “rewrite” tag informs the system of overriding block. <product_list> tag directs to app/code/core/Mage/Catalog/Product/List.php and this tag contains the class name used to override block class.

Next, we create a file *app/code/local/Magestore/Lesson18/Block/Catalog/Product/List.php* that contains class *Magestore_Lesson18_Block_Catalog_Product_List* that we have just declared. In reality, we can place this file anywhere in folder Block of Module, but we should use structure file/folder as the structure file/folder in Magento core.

Class Magestore_Lesson18_Block_Catalog_Product_List will extend class Mage_Catalog_Block_Product_List. The purpose of this is to help reuse the functions of system we do not rewrite.

```
class Magestore_Lesson18_Block_Catalog_Product_List extends
Mage_Catalog_Block_Product_List{
    // function that need to rewrite
    protected function _getProductCollection(){
        parent::_getProductCollection();
        //Define discount percent
        $percentDiscount = 20;
        foreach($this->_productCollection as $product)
        {
            $price = $product->getPrice();
                    $finalPriceNow = $product-
>getData('final_price');

                    $specialPrice = $price - $price *
$percentDiscount / 100;

                    // if special price is negative - negate the
discount - this may be a mistake in data
                    if ($specialPrice < 0)
                      $specialPrice = $finalPriceNow;

                    if ($specialPrice < $finalPriceNow)
                      $product->setFinalPrice($specialPrice); // set
the product final price
        }
        return $this->_productCollection;
    }
}
```

The result is shown as below:



Nokia 2610 Phone

3 Review(s) | Add Your Review

Regular Price: ~~$149.99~~ Add to Cart

Special Price: $119.99

The words "entry level" no longer mean "low-end," especially when it comes to the Nokia 2610. Offering advanced media and calling features without breaking the bank Learn More

Add to Wishlist | Add to Compare

BlackBerry 8100 Pearl

1 Review(s) | Add Your Review

Regular Price: ~~$349.99~~ Add to Cart

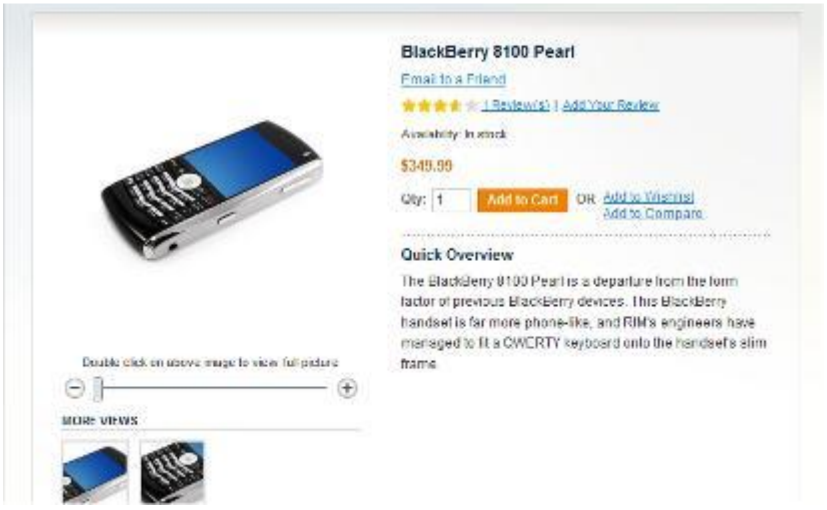
Special Price: $279.99

The BlackBerry 8100 Pearl is a departure from the form factor of previous BlackBerry devices. This BlackBerry handset is far more phone-like, and RIM's engineers have managed to fit a QWERTY keyboard onto the handset's slim frame. Learn More

Add to Wishlist | Add to Compare

2.2. Overriding Model Class

In the above section, we succeed in showing the special price in the product list page. However, it fails to show in the product detailed page.



BlackBerry 8100 Pearl

Email to a Friend

1 Review(s) | Add Your Review

Availability: In stock

$349.99

Qty: 1 Add to Cart OR Add to Wishlist Add to Compare

Quick Overview

The BlackBerry 8100 Pearl is a departure from the form factor of previous BlackBerry devices. This BlackBerry handset is far more phone-like, and RIM's engineers have managed to fit a QWERTY keyboard onto the handset's slim frame.

To solve this problem, override model class.

Overriding model class and overriding block class have many things in common. The following steps give you the instructions on how to override block class:

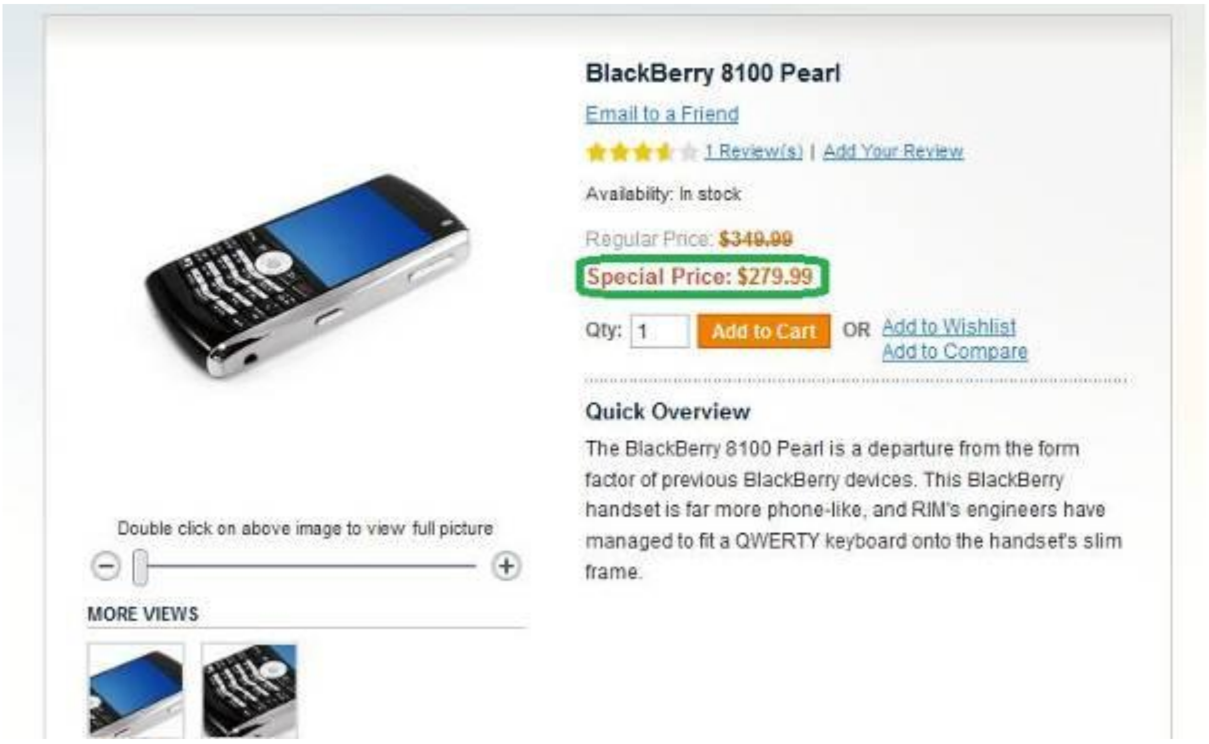
Step 1: Declare in file config.xml

```
<config>
  <global>
      <models>
      <lesson18>
      <class>Magestore_Lesson18_Model</class>
      </lesson18>
      <catalog>
        <rewrite>
            <product_type_price>
            Magestore_Lesson18_Model_Product_Type_Price
            </product_tye_price>
        </rewrite>
      </catalog>
      </models>
  </global>
</config>
```

Step 2: Create file *Magestore/Lesson18/Model/Product/Type/Price.php* which contains class Magestore_Lesson18_Model_Product_Type_Price to override as declared in file config:

```
class Magestore_Lesson18_Model_Product_Type_Price extends
Mage_Catalog_Model_Product_Type_Price
{
    public function getFinalPrice($qty=null, $product){
        if (is_null($qty) && !is_null($product-
>getCalculatedFinalPrice())) {
            return $product->getCalculatedFinalPrice();
        }

        $finalPrice = $this->getBasePrice($product, $qty);
        $product->setFinalPrice($finalPrice);

        Mage::dispatchEvent('catalog_product_get_final_price',
array('product' => $product, 'qty' => $qty));

        $finalPrice = $product->getData('final_price');
        $finalPrice = $this->_applyOptionsPrice($product, $qty,
$finalPrice);
        $finalPrice = max(0, $finalPrice);
      //Define discount percent
        $percentDiscount = 20;
      //Set final price for product
        $finalPrice = $finalPrice * (1-$percentDiscount/100);
        $product->setFinalPrice($finalPrice);

        return $finalPrice;
    }
}
```

The result is shown as below:


BlackBerry 8100 Pearl

Email to a Friend

1 Review(s) | Add Your Review

Availability: In stock

Regular Price: ~~$349.99~~

Special Price: $279.99

Qty: 1 Add to Cart OR Add to Wishlist Add to Compare

Quick Overview

The BlackBerry 8100 Pearl is a departure from the form factor of previous BlackBerry devices. This BlackBerry handset is far more phone-like, and RIM's engineers have managed to fit a QWERTY keyboard onto the handset's slim frame.

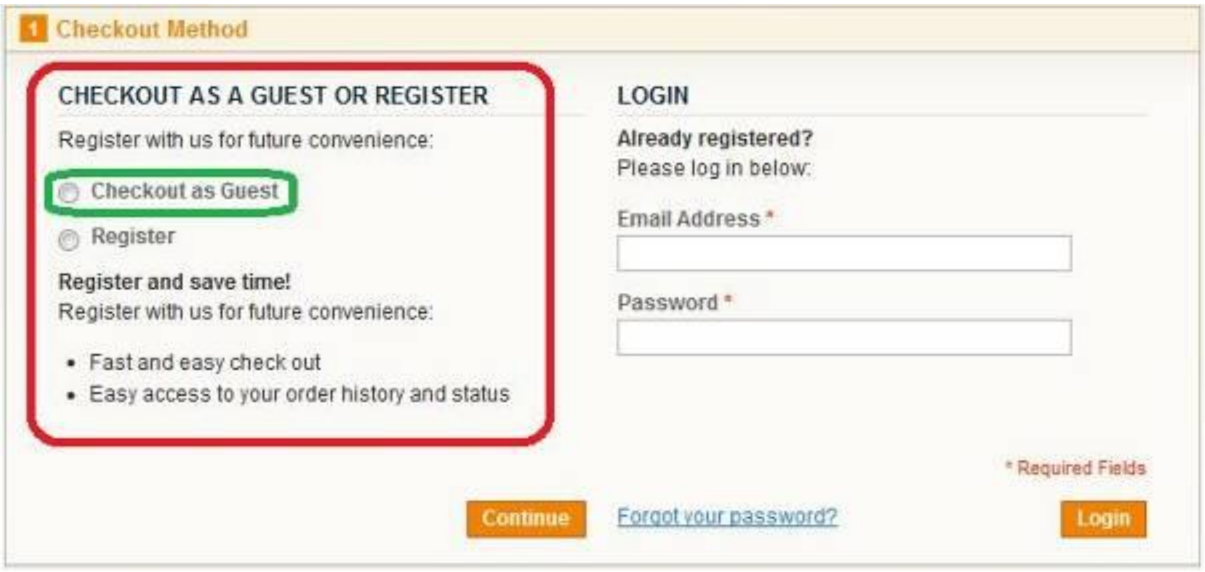
Double click on above image to view full picture

MORE VIEWS

2.3. Overriding Helper Class

With the above sale-off project, only customers who have account on your site can buy products. People who visit site as guests are not allowed to check out. We override helper class Mage_Checkout_Helper_Data.

Checkout method form before being overridden:


1 Checkout Method

CHECKOUT AS A GUEST OR REGISTER

Register with us for future convenience:

Checkout as Guest

Register

Register and save time!

Register with us for future convenience:

- Fast and easy check out
- Easy access to your order history and status

LOGIN

Already registered?

Please log in below:

Email Address *

Password *

* Required Fields

Continue Forgot your password? Login

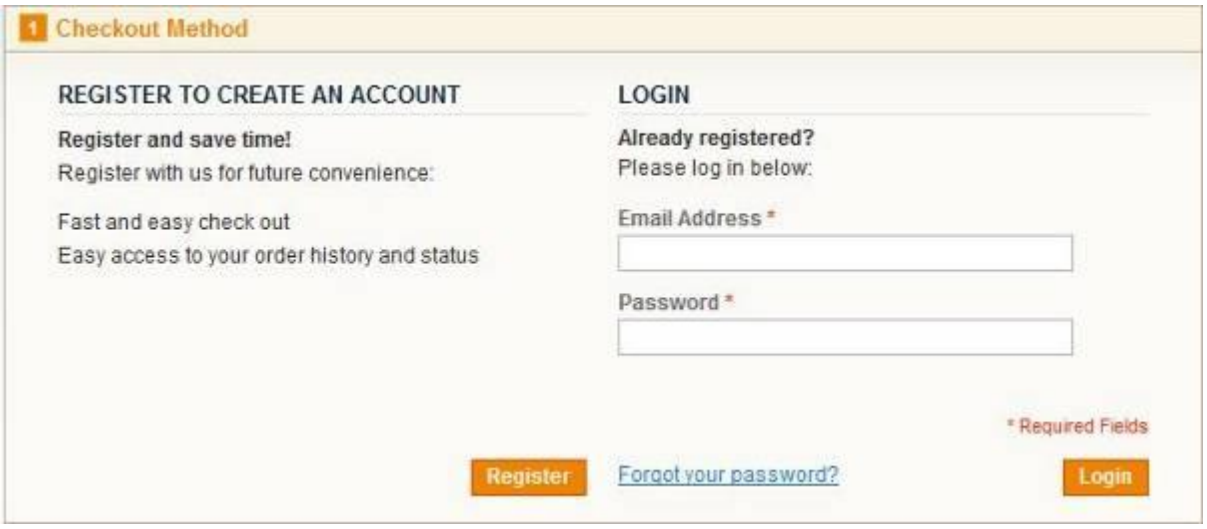
Like declaration of overriding block class and model class, we will declare helper class Mage_Checkout_Helper_Data as follows:

```
<config>
  <global>
      <helpers>
         <lesson18>
            <class>Magestore_Lesson18_Model</class>
         </lesson18>
         <checkout>
            <rewrite>
              <data>Magestore_Lesson18_Helper_Checkout_Data</data>
            </rewrite>
         </checkout>
      </helpers>
   </global>
</config>
```

Override helper class:

```
class Magestore_Lesson18_Helper_Checkout_Data extends
Mage_Checkout_Helper_Data
{
    public function isAllowedGuestCheckout(Mage_Sales_Model_Quote
$quote, $store = null)
    {
        return false;
    }
}
```

Here is the result:



2.4. Overriding Resource Model

Overriding resource model class is slightly different from the three classes mentioned. The difference is in file config.xml. A resource model is put in folder “Mysq14” and declared in file config.xml using tag <resourceModel></resourceModel>.

Instead of declaring only <catalog> tag before <rewrite> tag, you need to declare the whole link to folder mysq14.

Suppose that we want to override class Mage_Review_Model_Resource_Review_Collection, we will declare in file config.xml as follows:

```
<config>
  <global>
      <models>
         <lesson18>
               <class> Magestore_Lesson18_Model</class>
         </lesson18>
         <review_resource >
            <rewrite>
               <review_collection>
      Magestore_Lesson18_Model_Review_Resource_Review_Collection
               </review_collection>
            </rewrite>
         </review_resource >
      </models>
   </global>
</config>
```

Then, you override class in the same way as overriding class in model:

```
class Magestore_Lesson18_Model_Review_Resource_Review extends
Mage_Review_Model_Resource_Review_Collection {
    protected function _initSelect() {
        parent::_initSelect();
        $this->getSelect()
            ->join(array('detail' => $this->_reviewDetailTable),
'main_table.review_id = detail.review_id', array('title', 'detail',
'nickname', 'size'));
        return $this;
    }
}
```

III. SUMMARY

After learning this lesson, we should know:

- Overriding principle in Magento
- How to override in Magento

Back to top

###

Thank you for reading this e-book and hope that is is useful for you!

In the Volume 2 of the e-book ***"Magento Made Easy: Comprehensive Guide to Magento Setup and Development"***, we will mention more complicated Magento topics, so let's start to practice and wait for it.