LEARNING magento

#magento

Table of Contents

About

Chapter 1: Getting started with magento

Remarks

Magento is an open-source e-commerce platform written in PHP; a highly customizable e-commerce platform and content management system that can be used to build online stores for selling merchandise.

It provides common e-commerce features, such as shopping carts and inventory management, and encourages extensive customization to meet organization’s specific goals.

Magento is also an object-oriented PHP Framework that can be used to develop modern, dynamic web applications that tap into Magento's eCommerce features.

The major characteristics of Magento platform are:

- expandability
- scalability
- flexibility
- customizability
- open-source

Versions

Community Edition

Version	Release Date
1.9	2014-05-14
1.8	2013-12-11
1.7	2012-04-24
1.6	2011-08-08
1.5	2011-02-08
1.4	2010-02-12
1.3	2009-03-30
1.2	2008-12-29
1.0	2008-03-31

Enterprise Edition

Version	Release Date
1.14	2014-05-14
1.13	2013-10-11
1.12	2012-04-24
1.11	2011-08-08
1.10	2011-02-08
1.9	2010-07-19
1.8	2010-04-14
1.7	2010-01-19
1.6	2009-10-20
1.3	2009-04-15

Examples

Installation and Setup

Prerequisites and Requirements for Magento Community Edition 1.9

Hosting

- Apache 2.x (with mod_rewrite) or Nginx 1.7.x
- Due to the demands of processing Magento operations, it is recommended that you install Magento on a server with at least 2 GB of RAM. This will ensure that all of the software involved in managing the store will have enough memory to work.
- Ability to run scheduled jobs (crontab) with PHP 5.
- Ability to override options in .htaccess files.

PHP

- PHP 5.4, PHP 5.5
- Required extensions: PDO_MySQL, simplexml, mcrypt, hash, GD, DOM, iconv, curl, SOAP (for Webservices API)
- memory_limit no less than 256 MB (512 MB recommended)

Database

- MySQL 5.6 (Oracle, Percona, MariaDB)

SSL

- A valid security certificate is required for HTTPS.
- Self-signed SSL certificates are not supported

Installation:

Download and Set Up Magento Files

We are using openMage mirror as direct download for 1.9.2.4 branch is disabled and magento website require account. But you are encouraged to download copy from https://www.magentocommerce.com/download

```
cd /var/www/html
wget https://github.com/OpenMage/magento-mirror/archive/magento-1.9.zip
unzip magento-1.9.zip
rm magento-1.9.zip
rsync -avP magento-mirror-magento-1.9/. .
rm magento-mirror-magento-1.9 -r
sudo chown -R www-data:www-data /var/www/html/
chmod -R 0777 media var
```

Create a MySQL Database and User

access mysql console

```
mysql -u root -p
```

in mysql console

```
CREATE DATABASE magento;
CREATE USER magento_db_user@localhost IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON magento.* TO magento_db_user@localhost IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
exit
```

Complete the installation through the web interface

To access the web interface with your browser, navigate to your server's domain name or public

IP address:

```
http://domain_name/
```

Then follow on screen instructions

Troubleshooting Common Problems

Only the homepage works, all other pages return 404

Make sure `mod_rewrite` module has been installed in Apache and been enabled to load. See **step 2** for info on how to do this here: https://www.digitalocean.com/community/tutorials/how-to-set-up-mod_rewrite-for-apache-on-ubuntu-14-04

Make sure the you allow changes in the .htaccess by enabling it in your site conf. See **step 3**: https://www.digitalocean.com/community/tutorials/how-to-set-up-mod_rewrite-for-apache-on-ubuntu-14-04

Your `.htaccess` file may be mis-configured or missing: head over to Magento's download page: https://www.magentocommerce.com/download - download the relevant version and extract the .htaccess file to be placed in your Magento installation root.

Site works but not styles or scripts are loading

Make sure you have set the relevant permissions and ownership: **See here** for more info - http://devdocs.magento.com/guides/m1x/install/installer-privileges_before.html

Common solution: Try reindexing and flushing cache manually (in case the admin is too hard to navigate). Reindex via the command line: https://www.atwix.com/magento/process-magento-indexes-from-command-line/ Flush cache (via admin or command line): https://www.properhost.com/support/kb/23/How-To-Clear-The-Magento-Cache

Followed

Read Getting started with magento online: https://riptutorial.com/magento/topic/812/getting-started-with-magento

Chapter 2: Add different price for multiple store using Magento SOAP API

Examples

Magento SOAP V1

You need to change price scope **'Global' to 'website'** (Sysytem->Configuration->Catalog->Catalog->Price)

```
$client = new SoapClient('http://your-web-site/api/soap/?wsdl');
$API_USER = 'your-api-user';
$API_KEY = 'your-api-key';
$session = $client->login($API_USER, $API_KEY);
$result = $client->call($session, 'catalog_product.update', array('test-product',
array('price' => '100'),'your-store-code'));
print "<pre>";
print_r($result);
print "</pre>";
```

Read Add different price for multiple store using Magento SOAP API online: https://riptutorial.com/magento/topic/7410/add-different-price-for-multiple-store-using-magento-soap-api

Chapter 3: Collections

Examples

Get Model Collections

```
// get existing collections
$orders = Mage::getModel('sales/order')->getCollection();
$products = Mage::getModel('catalog/product')->getCollection();
$customers = Mage::getModel('customer/customer')->getCollection();
```

Get Additional Object Attributes

```
// $orders is collection
$orders->addAttributeToSelect('status'); // get status attribute
$orders->addAttributeToSelect('*'); // get all attributes
```

Filtering

```
// filter by creation date
$date = new Zend_Date();
$toDate = $date->get(Zend_Date::W3C); // today
$fromDate = $date->sub('1', Zend_Date::MONTH)->get(Zend_Date::W3C); // one month ago

$orders->addAttributeToFilter('created_at', array('from' => $fromDate, 'to' => $toDate));

// more filtering, type AND
$orders->addAttributeToFilter('status', 'pending');
$orders->addAttributeToFilter('state', array('nlike' => 'new'));

// more filtering, type OR
$processing = array('eq' => 'processing');
$complete = array('like' => 'complete');
$orders->addAttributeToFilter('status', array($processing, $complete));
```

Sorting

```
// sort by creation date
$orders->setOrder('created_at', 'asc');
$orders->setOrder('created_at', 'desc');
$orders->setOrder('created_at'); // default direction is 'desc'
```

Access

```
// iterating over items in collection
foreach ($orders as $singleOrder) {
    // do something with individual objects
    var_dump($singleOrder->getData());
}
```

```
// get first/last item in collection
$first = $orders->getFirstItem();
$last = $orders->getLastItem();
```

Get Collection Object

```
    // get product collection object
    $productCollection = Mage::getResourceModel('catalog/product_collection');
    // get customer collection object
    $customerCollection = Mage::getResourceModel('customer/customer_collection');
    // get order collection object
    $orderCollection = Mage::getResourceModel('sales/order_collection');
```

Read Collections online: https://riptutorial.com/magento/topic/5937/collections

Chapter 4: Collections

Examples

Product Collection

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

Selecting the specific Attribute

```
$ProductCollection->addAttributeToSelect(array('name', 'product_url', 'small_image'));
```

Selecting the All Attribute

```
$ProductCollection->addAttributeToSelect('*');
```

Add Filter on Collection

```
$ProductCollection->addFieldToFilter('is_active',1);
```

Add Product Attribute Filter on Collection

```
$ProductCollection->addAttributeToFilter('weight', array('gt' => 100));
```

Set Order

```
$ProductCollection->setOrder('id','ASC');
```

Set limit

```
$ProductCollection->setPageSize(10);
```

Set Current Page

```
$ProductCollection->setCurPage(1);
```

Category Collection of a specific store and specific level

```
$rootId     = Mage::app()->getStore($storeId)->getRootCategoryId();
$categories = Mage::getModel('catalog/category')->getCollection()
    ->addAttributeToSelect('*')
    ->addFieldToFilter('path', array('like'=> "1/$rootId/%"))
    ->addAttributeToFilter('level', 2)
    ->addAttributeToFilter('is_active', 1);
```

Read Collections online: https://riptutorial.com/magento/topic/6763/collections

Chapter 5: Collections

Examples

Product Collection

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

Selecting the specific Attribute

```
$productCollection->addAttributeToSelect(array('name', 'product_url', 'small_image'));
```

Selecting the All Attributes

```
$productCollection->addAttributeToSelect('*');
```

Add Filter on Collection

```
$productCollection->addFieldToFilter('is_active', 1);
```

Set Order

```
$productCollection->setOrder('id', 'ASC');
```

Set limit

```
$productCollection->setPageSize(10);
```

Set Current Page

```
$productCollection->setCurPage($page);
```

Read Collections online: https://riptutorial.com/magento/topic/7206/collections

Chapter 6: Complete Product Collection

Introduction

Complete filters for product collection based on visibility,store,And OR, stock status, status, etc

Examples

Filtering product collection

```
$model = Mage::getModel('catalog/product')->getCollection()
```

Filter based on store:

```
$mode->addStoreFilter($storeId)
```

Filter based on product type:

```
$mode->addAttributeToFilter('type_id', 'configurable')
$mode->addAttributeToFilter('type_id', 'simple')
```

Filter based on status:

```
$model->addAttributeToFilter('status',Mage_Catalog_Model_Product_Status::STATUS_DISABLED)
$model->addAttributeToFilter('status',Mage_Catalog_Model_Product_Status::STATUS_ENABLED)
```

Filter using null and notnull:

```
$model->addAttributeToFilter('short_description', array('null' => true))
$model->addAttributeToFilter('short_description', array('notnull' => true))
```

Filter using greater than and less than:

```
$model->addAttributeToFilter('entity_id', array('gt' => 64230))
$model->addAttributeToFilter('entity_id', array('lt' => 64230))
```

Filter using greater than and equal to:

```
$model->addAttributeToFilter('entity_id', array('gteq' => 64230))
```

Filter using less than and equal to:

```
$model->addAttributeToFilter('entity_id', array('lteq' => 64230))
```

Filter using in and not in:

```
$model->addAttributeToFilter('entity_id', array('in' => array(1,4,64231)))
$model->addAttributeToFilter('entity_id', array('nin' => array(1,4,64231)))
```

Filter products by a range of entity id:

```
$model->addAttributeToFilter('entity_id', array(
             'from' => 64229,
             'to' => 64231
             ))
```

Filter based on product visibility:

```
$model->addAttributeToFilter('visibility', 4) //catalog,search
$model->addAttributeToFilter('visibility', 3) //catalog
$model->addAttributeToFilter('visibility', 2) //search
$model->addAttributeToFilter('visibility', 1) //not visible individually
```

Filter using like and not like:

```
$model->addAttributeToFilter('sku', array('nlike' => '5713%'))
$model->addAttributeToFilter('sku', array('like' => '%shirt%'))
```

Filter using equal to and not equal to:

```
$model->addAttributeToFilter('sku', array('neq' => 'shirt'))
$model->addAttributeToFilter('sku', array('eq' => 'shirt'))
```

Filter in stock products:

```
$model->joinField('is_in_stock',
                'cataloginventory/stock_item',
                'is_in_stock',
                'product_id=entity_id',
                'is_in_stock=1', //make this 0 for out of stock products
                '{{table}}.stock_id=1',
                'left')
```

Set order by:

```
$model->setOrder('entity_id','desc')
```

Set Page Size:

```
$model->setPageSize(100)
```

Read Complete Product Collection online: https://riptutorial.com/magento/topic/9261/complete-product-collection

Chapter 7: Create Enterprise Gift Cards Programmatically

Examples

Create Single Gift Card

```
// Instantiate Gift Card Model
$gift_card = Mage::getModel('enterprise_giftcardaccount/giftcardaccount');

// Populate Gift Card values
$gift_card
    // Your redeemable code, doesn't have to be in this format.
    ->setCode('2i2j2j-24k1ii1-67774k-231l')
    // Also has STATUS_DISABLED
    ->setStatus($gift_card::STATUS_ENABLED)
    // YYYY-MM-DD format date
    ->setDateExpires('2015-04-15')
    ->setWebsiteId(1)
    // Also has STATE_USED, STATE_REDEEMED, and STATE_EXPIRED
    ->setState($gift_card::STATE_AVAILABLE)
    // Also has NOT_REDEEMABLE value.
    ->setIsRedeemable($gift_card::REDEEMABLE)
    // Int or float (or String that can be parsed into either) currency amount.
    ->setBalance(25);

// Save the fleshed out gift card.
$gift_card->save();

// Can use the gift card for further use, return it, or return it's ID
return $gift_card // ->getId();
```

Read Create Enterprise Gift Cards Programmatically online:
https://riptutorial.com/magento/topic/2387/create-enterprise-gift-cards-programmatically

Chapter 8: Current url

Syntax

- `$this->helper('core/url')->getCurrentUrl();`

Examples

Homepage

return : http://www.example.com/

Product page

return : http://www.example.com/my-product.html

Check if current url is secure

```
$isSecure = Mage::app()->getStore()->isCurrentlySecure();
```

This will return true if the current url is secure.

Read Current url online: https://riptutorial.com/magento/topic/2150/current-url

Chapter 9: Custom Attributes

Introduction

Custom Attributes for sales, category, etc.

Examples

Sales attribute

Custom attribute in sales related tables like: sales_flat_quote, sales_flat_order_item, sales_flat_order, etc table

In your installation file sql/some_setup/mysql-install-0.1.0.php:

```
<?php
 $installer = $this;
 $installer->startSetup();
 $installer->addAttribute('quote', 'custom_field', array('type' => 'varchar'));
 $installer->addAttribute('order', 'custom_field', array('type' => 'varchar'));
 $installer->endSetup();
 ?>
```

Another way of doing it is:

```
<?php
$installer = $this;
$installer->startSetup();
$installer->run("ALTER TABLE sales_flat_order_item ADD COLUMN 'custom_field' DECIMAL(12,4)
NULL;");
$installer->endSetup();
?>
```

Make sure to clear cache after this.

Read Custom Attributes online: https://riptutorial.com/magento/topic/9267/custom-attributes

Chapter 10: EAV (Entity Attribute Value)

Remarks

Entity

Stores information about the type of the data being stored. In the case of Magento this is customer, product, category, etc.

Attribute

The individual properties of each of the entities, e.g. name, weight, email address etc.

Value

The value of a given entity and attribute. For example, we may specify the customer entity and the email attribute and then give it the value hello@example.com.

Database Schema

eav_entity

The entity table.

eav_entity_attribute

The attribute table.

eav_entity_{type}

The values tables. Types are datetime, decimals, int, text and varchar.

Important to note that eav_entity_varchar table has the type varchar for values even if date or integer would suit the value better.

Models Versus Resource Models

All the models inside Mage/Eav/Model/Resource are Mysql4 and are resource models.

In addition Entity/Abstract.php and Entity/Setup.php.

Flat Versus EAV

The EAV models are more complex, providing logic to save and load from multiple tables, whereas the flat or standard models are relatively straightforward (traditional).

Standard models mainly manage their properties with data setters and getters working with a single table. EAV models mainly manage their attribute models. Standard models only save their data to a table and load from it. EAV models load all (or a specific set) attributes after loading base data and save attributes after saving data (including inserting, updating and deleting attributes).

EAV Resource Model Examples

To find entities using the EAV storage schema searching for the string extends Mage_Eav_Model_Entity_Abstract can be used. This should reveal all resource models based on the EAV structure. However, the resulting list will include many obsolete classes from the Mage_Sales module that no longer are being used.

The only modules containing entities using the EAV storage schema are Mage_Catalog (categories and products) and Mage_Customer (customers and addresses).

Customer Groups use the flat table storage schema. All the sales entities where converted to flat table entities with the release of Magento 1.4.

The reason that EAV is used is so that entities can have an undetermined number of properties and therefore remain flexible. For example, when you add a new attribute to a Customer entity (which is an EAV entity), the database table does not need to be altered for this new attribute to be added.

Advantages

Flexibility Database schema doesn’t need to change with the model Quick to implement

Disadvantages

- Inefficient

A query returning 20 columns normally would consist of 20 self joins in EAV. However, the Mage_Eav module generally does not use joins to load the attribute value data. Instead union selects are used. Joins are only used for filtering EAV collections.

- No mechanism for relationships between subtypes
- No grouping of entity subtypes

Website and Store Scopes

To handle website and store scope attribute values within EAV, a store_id value exists on the catalog entity to show scope which links back to core_store. Along with the normal stores (store views) there is also a store ‘0’ which is the global value. When on a particular store, the system will first check for an entity value on the current store and then fall back to the global entity. Mage_Customer EAV entities do not have a store_id scope column.

Insert, Update and Delete

To determine if an insert, update or delete needs to be performed on an attribute, a comparison is made against the original object. The original object is basically a duplicate of the data object when the entity was retrieved from the database.

- If the attribute exist originally and its new value is not empty; it updates.

- If the attribute exist originally but its new value is set to empty; it deletes. - If the attribute doesn’t exist originally and its new value is not empty; it inserts.

Attribute Management

Attribute Models

Attribute Model *Represents the attribute in the database form, its logic is standard across all attributes and is difficult to change*.

Frontend Model

The attribute’s interface to the frontend and provides any logic that the attribute requires on the frontend, e.g. the getUrl() method on images.

Backend Model

These perform validation on the attribute before it is saved to the database. For example, the password backend model converts the password into a hash before it is saved. It also checks that the password and password confirmation match before saving.

Source Models

Used to populate the options available for an attribute, e.g. catalog/product_status has enabled and disabled.

Required Methods

A source model requires:

```
<?php
    public function getAllOptions();
    public function getOptionText($value);
?>
```

Usually only getAllOptions() needs to be implemented though since an implementation for getOptionText() already exists in the abstract source model Mage_Eav_Model_Entity_Attribute_Source_Abstract.

A frontend model does not require the method getValue().

A backend model requires:

```
<?php
    public function getTable();
    public function isStatic();
    public function getType();
    public function getEntityIdField();
    public function setValueId($valueId);
    public function getValueId();
    public function afterLoad($object);
    public function beforeSave($object);
    public function afterSave($object);
    public function beforeDelete($object);
    public function afterDelete($object);
    public function getEntityValueId($entity);
    public function setEntityValidId($entity, $valueId);
?>
```

All these methods are implemented in the abstract backend model Mage_Eav_Model_Entity_Attribute_Backend_Abstract. For custom backend models only the methods requiring customisation need to be overridden.

System Configuration Source Models

Cannot be used for EAV attributes. EAV source models implement the getAllOptions method while adminhtml source models implement the toOptionArray() method.

Default system configuration source models can be found in Mage/Adminhtml/Model/System/Config/Source/.

Attribute Source Models

The purpose of Attribute Source Models is to supply the list of options and values for select and multiselect attributes. They also supply the column information to the catalog flat table indexer if required.

To get a list of all options for an attribute, perform the following:

```
<?php
    $options = $attribute->getSource()->getAllOptions(false);

    // or for admin
    $options = $_attribute->getSource()->getAllOptions(true, true);
?>
```

Default Attribute Models

If no class is specified as a frontend, backend or - for select or multiselect attributes - source models, a default class is used.

The default attribute frontend model is Mage_Eav_Model_Entity_Attribute_Frontend_Default.

The default attribute backend model depends on the attribute code and is determined in the method Mage_Eav_Model_Entity_Attribute::_getDefaultBackendModel().

```
<?php
    protected function _getDefaultBackendModel()
    {
        switch ($this->getAttributeCode()) {
            case 'created_at':
                return 'eav/entity_attribute_backend_time_created';

            case 'updated_at':
                return 'eav/entity_attribute_backend_time_updated';
            case 'store_id':
                return 'eav/entity_attribute_backend_store';

            case 'increment_id':
                return 'eav/entity_attribute_backend_increment';
        }

        return parent::_getDefaultBackendModel();
    }
?>
```

If the method falls through to the last line Mage_Eav_Model_Entity_Attribute_Backend_Default is used.

The default source model is set in Mage_Eav_Model_Entity_Attribute_Source_Table. This is set in the catalog modules attribute model. The default config source model specified in the eav module is never used.

Add Attribute

To add EAV attributes, use Mage_Eav_Model_Entity_Setup by extending in the setup class.

addAttribute() Creates the attributes, add it to groups and sets (including default) , or updates if it already exists updateAttribute() Updates the attribute data only. Custom setup classes can be used to extend these methods, adding additional data or simplifying the arguments needed.

Flat Tables

Flat catalog attributes are managed by indexers:

Mage_Catalog_Model_Resource_Product_Flat_Indexer::updateAttribute()
Mage_Catalog_Model_Resource_Category_Flat::synchronise() Product attributes get added to the flat table if they are (see
Mage_Catalog_Model_Resource_Product_Flat_Indexer::getAttributeCodes()):

Static (backend type) Filterable Used in product listing Used for promo rules Used for sort by System Attributes There is a different flat table for each store, each one contains a different store-scoped entity attribute value. Multi-lingual values are managed by having different stores for each language.

Examples

implement the interface of attribute frontend, source, and backend models

Frontend Interface

```
/**
* Entity attribute frontend interface
*
* Frontend is providing the user interface for the attribute
```

```
*
*/
interface Mage_Eav_Model_Entity_Attribute_Frontend_Interface
{

}
```

Source Interface

```
/**
* Entity attribute select source interface
*
* Source is providing the selection options for user interface
*
*/
interface Mage_Eav_Model_Entity_Attribute_Source_Interface
{
/**
* Retrieve All options
*
* @return array
*/
public function getAllOptions();

/**
* Retrieve Option value text
*
* @param string $value
* @return mixed
*/
public function getOptionText($value);
}
```

Backend Interface

```
/**
* Entity attribute backend interface
*
* Backend is responsible for saving the values of the attribute
* and performing pre and post actions
*
*/
interface Mage_Eav_Model_Entity_Attribute_Backend_Interface
{
public function getTable();
public function isStatic();
public function getType();
public function getEntityIdField();
public function setValueId($valueId);
public function getValueId();
public function afterLoad($object);
public function beforeSave($object);
public function afterSave($object);
public function beforeDelete($object);
public function afterDelete($object);

/**
* Get entity value id
*
* @param Varien_Object $entity
*/
public function getEntityValueId($entity);

/**
* Set entity value id
*
* @param Varien_Object $entity
* @param int $valueId
*/
public function setEntityValueId($entity, $valueId);
}
```

Read EAV (Entity Attribute Value) online: https://riptutorial.com/magento/topic/7121/eav--entity-attribute-value-

Chapter 11: Get category name from product page

Examples

Get the parent category

```
$_cat = new Mage_Catalog_Block_Navigation();
$curent_cat = $_cat->getCurrentCategory();
$curent_cat_id = $curent_cat->getId();
$parentId=Mage::getModel('catalog/category')->load($curent_cat_id)->getParentId();
$parent = Mage::getModel('catalog/category')->load($parentId);
$categorydaddy = $parent->getName();
```

Get the current category

```
    $categoryName = Mage::registry('current_category')->getName();
    foreach ($categoryName as $_category):
       $categoryName = $_category->getName();
    endforeach;
```

Read Get category name from product page online: https://riptutorial.com/magento/topic/6078/get-category-name-from-product-page

Chapter 12: Get current User

Examples

Get current Admin User

```
Mage::getSingleton('admin/session')->getUser();
```

Get current Customer

```
Mage::helper('customer')->getCustomer();
```

or

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

Check if user is logged in

```
Mage::getSingleton('customer/session')->isLoggedIn()
```

Read Get current User online: https://riptutorial.com/magento/topic/2157/get-current-user

Chapter 13: Get Products from Database

Examples

Get product by sku

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

Get product by ID

```
$id = 1;
$product = Mage::getModel('catalog/product')->load($id);
if($product->getId()){
    //product was found
}
```

Product collection - LIKE query

```
$collection = Mage::getModel('catalog/product')->getCollection();
$collection->addAttributeToFilter('sku', array('like' => 'UX%'));
```

Get product collection by attribute

```
$collection = Mage::getModel('catalog/product')->getCollection();
// Using operator
$collection->addAttributeToFilter('status', array('eq' => 1));
// Without operator (automatically uses 'equal' operator
$collection->addAttributeToFilter('status', 1);
```

Get data from product object

```
// First load a product object

$product->getSku();
$product->getName();

// Alternative method
$product->getData('sku');
$product->getData('name');
```

Get data form product collection

```
// First load a collection object

foreach($collection as $product) {
    $product->getSku();
    $product->getName();

    // Alternative method
    $product->getData('sku');
    $product->getData('name');
}
```

Product collection - with attributes

```
//all attributes
$collection = Mage::getModel('catalog/product')
    ->getCollection()
    ->addAttributeToSelect('*');
//specific attributes
$collection = Mage::getModel('catalog/product')
    ->getCollection()
    ->addAttributeToSelect('name');
//certain attributes are special, such as price and images
//for images, then you can use 'getMediaGalleryImages'
$product->load('media_galley');
```

Check if the product was correctly loaded

```
$productFound = ($product->getId() !== null)
```

Get product ID by SKU

```
$sku = 'some-sku';
$productId = Mage::getModel('catalog/product')->getIdBySku($sku);
if($productId){
   //sku exists
}
```

Get product collection from a list of SKUs

`$skuList = array('SKU-1', 'SKU-2',...,'SKU-n);`

```
$_productCollection = Mage::getModel('catalog/product')
->getCollection()
->addAttributeToFilter('sku', array('in' => $skuList));
```

OR

```
$_productCollection = Mage::getResourceModel('catalog/product_collection')
->addAttributeToFilter('sku', array('in' => $skuList));
```

Set Limit in product collection

```
$collection = Mage::getModel('catalog/product')
            ->getCollection()
            ->setPageSize(20)
            ->setCurPage(1);
```

Read Get Products from Database online: https://riptutorial.com/magento/topic/1102/get-products-from-database

Chapter 14: Get store name and other details from system configuration

Examples

Get the frontend name for the current store view

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

Get current store ID

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

Get current store code

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

This returns store code, e.g. 'en' for a storefront that is setup for English and called 'en', and not the numerical id.

Determine if store view is enabled

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

Get website ID for current store

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

Get the current store model

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

Returns an instance of `Mage_Core_Model_Store`

Get group name for store

```
Mage::app()->getStore()->getGroup()->getName()
```

Get all Magento stores

```
Mage::app()->getStores();
```

Returns an array of `Mage_Core_Model_Store` models.

Read Get store name and other details from system configuration online: https://riptutorial.com/magento/topic/1876/get-store-name-and-other-details-from-system-configuration

Chapter 15: Getting Magento URLs

Syntax

- $this->getSkinUrl('images/my-image.jpg');

Parameters

images path	details
example: *'images/my-images.jpg'*	path for image

Remarks

Get formatted images url and avoid theme dependences.

Examples

In current Interface/Theme

return http://www.example.com/skin/frontend/{interface}/{theme}/images/my-image.jpg

Get Skin Url

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

Get Base Url

```
Mage::getBaseUrl();
```

Secure Skin Url

```
$this->getSkinUrl('images/imagename.gif', array('_secure'=>true));
```

Get Media Url

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

Unsecure Skin Url

```
$this->getSkinUrl('images/imagename.jpg');
```

Get Store Url

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

Get Js Url

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

Get Current Url

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

Read Getting Magento URLs online: https://riptutorial.com/magento/topic/2148/getting-magento-urls

Chapter 16: Helpers

Examples

Creating a helper

Helpers should extend from `Mage_Core_Helper_Abstract`:

```
# File: app/code/local/Vendor/Package/Helper/Data.php
class Vendor_Package_Helper_Data extends Mage_Core_Helper_Abstract
{
    public function multiply($a, $b)
    {
        return $a * $b;
    }
}
```

To be able to access is via `Mage::helper` you need to define a helper alias in a `config.xml` file to allow the Magento autoloader to find your class:

```
<!-- File: app/code/local/Vendor/Package/etc/config.xml -->
<global>
    <helpers>
        <alias_here>
            <class>Vendor_Package_Helper</class>
        </alias_here>
    </helpers>
</global>
```

Assuming your module is correctly configured and you have cleared your cache, you should now be able to use your helper like so:

```
$result = Mage::helper('alias_here')->multiply(2, 4); // int(8)
```

Note: if you're using a Data class, its helper name is implied if you don't specify one. For example, the following two examples are identical:

```
Mage::helper('alias_here');
Mage::helper('alias_here/data');
```

Read Helpers online: https://riptutorial.com/magento/topic/5904/helpers

Chapter 17: How to Filter Collections

Parameters

Parameter	Details
$addFieldToFilter(**$field**, $condition = null)	{ string } The field we are adding to the filter.
$addFieldToFilter($field, **$condition = null**)	{ mixed } The definition of the filter we will use.
addAttributeToFilter(**$attr**, $condition = null, $join = 'inner')	{ string } The field we are adding to the filter.
addAttributeToFilter($attr, **$condition = null**, $join = 'inner')	{ mixed } The definition of the filter we will use.
addAttributeToFilter($attr, $condition = null, **$join = 'inner'**)	{ ('inner','left') } The type of sql join to use when joining the EAV table.

Remarks

Filter Comparison Arguments

Magento also offers a flexible way of filtering using comparison operators as well. Here is a list of valid operators and their syntax:

All comparison arguments can be passed to the second parameter of either the `addFieldToFielter()` or `addAttributeToFilter()` methods.

```
$collection_of_products->addAttributeToFilter('visible',array("eq"=>1));
```

Comparison	Argument Array	Resulting SQL Snippet
Equals	array("eq"=>$var)	WHERE (\`my_field\` = $var)
Not Equal	array("neq"=>$var)	WHERE (\`my_field\` != $var)
Like	array("like"=>$var)	WHERE (\`my_field\` LIKE $var)
Not Like	array("nlike"=>$var)	WHERE (\`my_field\` NOT LIKE $var)
Is	array("is"=>$var)	WHERE (\`my_field\` IS $var)
In	array("in"=>$var)	WHERE (\`my_field\` IN($var))

Comparison	Argument Array	Resulting SQL Snippet
Not In	array("nin"=>$var)	WHERE (`my_field` NOT IN($var))
Null	array("null"=>true)	WHERE (`my_field` IS NULL)
Not Null	array("notnull"=>true)	WHERE (`my_field` IS NOT NULL)
Greater Than	array("gt"=>$var)	WHERE (`my_field` > $var)
Less Than	array("lt"=>$var)	WHERE (`my_field` < $var)
Greater Than or Equal	array("gteq"=>$var)	WHERE (`my_field` >= $var)
Less Than or Equal	array("lteq"=>$var)	WHERE (`my_field` <= $var)
Find in Set	array("finset"=>array($var))	WHERE (find_in_set($var,`my_field`)
From and To	array("from"=>$var1, "to"=>$var2)	WHERE (`my_field` >= $var1 AND `my_field` <= $var2)

Examples

Filtering Collections

Magento has a powerful set of methods to filter collections. Since there are two types of Objects that can be contained in collections, we must first determine which type of data we are working with before we can filter it. Magento implements a EAV data model for entities such as products and categories. There is a different set of methods to use if we are filtering a collection of EAV Objects.

In Magento, Orders are not stored as EAV Objects. This makes the orders collection a good example for filtering a basic collection.

```
$collection_of_orders = Mage::getModel('sales/order')->getCollection();
$collection_of_orders->addFieldToFilter('status','processing');
```

If we look at the products collection, we can see that the products are stored in an EAV data model. We can easily filter by EAV attributes as well.

```
$collection_of_products = Mage::getModel('catalog/product')->getCollection();
$collection_of_products->addAttributeToFilter('visible',1);
```

Handling ANDs and ORs in Filters

When we query our data, we often need more than one filter to get the exact data set we are looking for. In SQL, we handle this with AND and OR clauses. We can achieve the same thing with collections.

To add an AND clause to your query, just simply add another method call. This will append the second filter to the original WHERE statement joining it with an AND.

```
Mage::getModel('catalog/product')->getCollection()
        ->addFieldToFilter('sku',array('like'=>'a%'))
        ->addFieldToFilter('sku',array('like'=>'%b'));
```

The resulting WHERE clause will look like this:

```
WHERE (e.sku like 'a%') AND (e.sku like '%b')
```

Now lets say we want all skus that start with 'a' OR end with 'b'. How do we add an OR clause? Thanks to Magento's collections, it is pretty straight forward. We add the filter as a second element in the filter array.

```
Mage::getModel('catalog/product')->getCollection()
        ->addFieldToFilter('sku', array(
            array('like'=>'a%'),
            array('like'=>'%b')
        ));
```

Now, the resulting WHERE clause will look like this:

```
WHERE (((e.sku like 'a%') or (e.sku like '%b')))
```

Read How to Filter Collections online: https://riptutorial.com/magento/topic/5849/how-to-filter-collections

Chapter 18: Logging to file

Syntax

- public static function log($message, $level = null, $file = '', $forceLog = false)

Parameters

Parameter	Details
string $message	The message that will be logged
integer $level	Log level
string $file	Path and name with extension of file that will be saved to `var/log/`. If NULL or not specified then `system.log` will be used.
bool $forceLog	If set to `TRUE` log will be written even though developer mode is off and logging is inactive.

Remarks

The Logging is turned off by default unless developer mode is active.

All exceptions are logged in `exceptions.log` no matter if logging is enabled in configuration.

Logging can be enabled by logging into Magento Admin and proceeding to:

- System > Configuration (top bar)
- Developer (left menu)
- Log Settings section
- Select Yes from `Enabled` dropdown list.
- Save Configuration in the right top corner.

Message variable type

Even though documentation defines that message should be a string, if an array is passed there's a code block in that method to take care of that with `print_r`:

```
if (is_array($message) || is_object($message)) {
    $message = print_r($message, true);
}
```

Log level

If the level parameter is set to null then DEBUG level is taken.

`$level = is_null($level) ? Zend_Log::DEBUG : $level;` The levels are declared in file: `lib\Zend\log.php`

```
const EMERG   = 0;  // Emergency: system is unusable
const ALERT   = 1;  // Alert: action must be taken immediately
const CRIT    = 2;  // Critical: critical conditions
const ERR     = 3;  // Error: error conditions
const WARN    = 4;  // Warning: warning conditions
const NOTICE  = 5;  // Notice: normal but significant condition
const INFO    = 6;  // Informational: informational messages
const DEBUG   = 7;  // Debug: debug messages
```

Constants in form of `Zend_Log::INFO` or integer number in range specified above can be passed as log level parameter.

Examples

Custom log file

```
Mage::log('My log entry', null, 'mylogfile.log');
```

This wil log to

```
/var/log/mylogfile.log
```

Default logging

```
Mage::log('My log entry');
Mage::log('My log message: '.$myVariable);
Mage::log($myArray);
Mage::log($myObject);
```

This will log to `/var/log/system.log`

Objects and Arrays are automatically written via a `print_r()` directive. Watch out when using objects since these can get substantial in size.

```
Mage::logException($e);
```

This will log exception trace string to `/var/log/exception.log`

Read Logging to file online: https://riptutorial.com/magento/topic/2363/logging-to-file

Chapter 19: Magento Caching

Examples

How to cache custom data into Magento

```
const CACHE_TAG_NAMESPACE_MODULE = "YOUR_MODULES_CACHE_TAGS";
$cacheGroup = 'namespace_module';
$useCache = Mage::app()->useCache($cacheGroup);
if (true === $useCache) {
    $cacheId = 'unique_name';
    if ($cacheContent = Mage::app()->loadCache($cacheId)) {
        $html = $cacheContent;
        return $html;
    } else {
        try {
            $cacheContent = $html;
            $tags = array(model::CACHE_TAG_NAMESPACE_MODULE);
            $lifetime = Mage::getStoreConfig('core/cache/lifetime');
            Mage::app()->saveCache($cacheContent, $cacheId, $tags, $lifetime);
        } catch (Exception $e) {
            // Exception = no caching
            Mage::logException($e);
        }
        return $html;
   }
}
// Default:
return $html;
```

Clean cache by cache ID

```
Mage::app()->removeCache($cacheId);
```

Flush all Magento cache entries

```
Mage::app()->cleanCache()
```

or:

```
Mage::app()->getCacheInstance()->flush();
```

Use Redis as a cache backend

Redis configuration:

1. Install redis (2.4+ required)
2. Install phpredis
3. Install the Magento extension `Cm_Cache_Backend_Redis` (only for Magento 1.7 and below)

4. Edit your `app/etc/local.xml`:

```
<global>
  ...
  <cache>
    <backend>Cm_Cache_Backend_Redis</backend>
    <backend_options>
      <server>127.0.0.1</server> <!-- or absolute path to unix socket -->
      <port>6379</port>
      <persistent></persistent>
      <database>0</database>
      <password></password>
      <force_standalone>0</force_standalone>
      <connect_retries>1</connect_retries>
      <automatic_cleaning_factor>0</automatic_cleaning_factor>
      <compress_data>1</compress_data>
      <compress_tags>1</compress_tags>
      <compress_threshold>20480</compress_threshold>
      <compression_lib>gzip</compression_lib> <!-- Supports gzip, lzf and snappy -->
    </backend_options>
  </cache>
  ...
</global>
```

Read Magento Caching online: https://riptutorial.com/magento/topic/4902/magento-caching

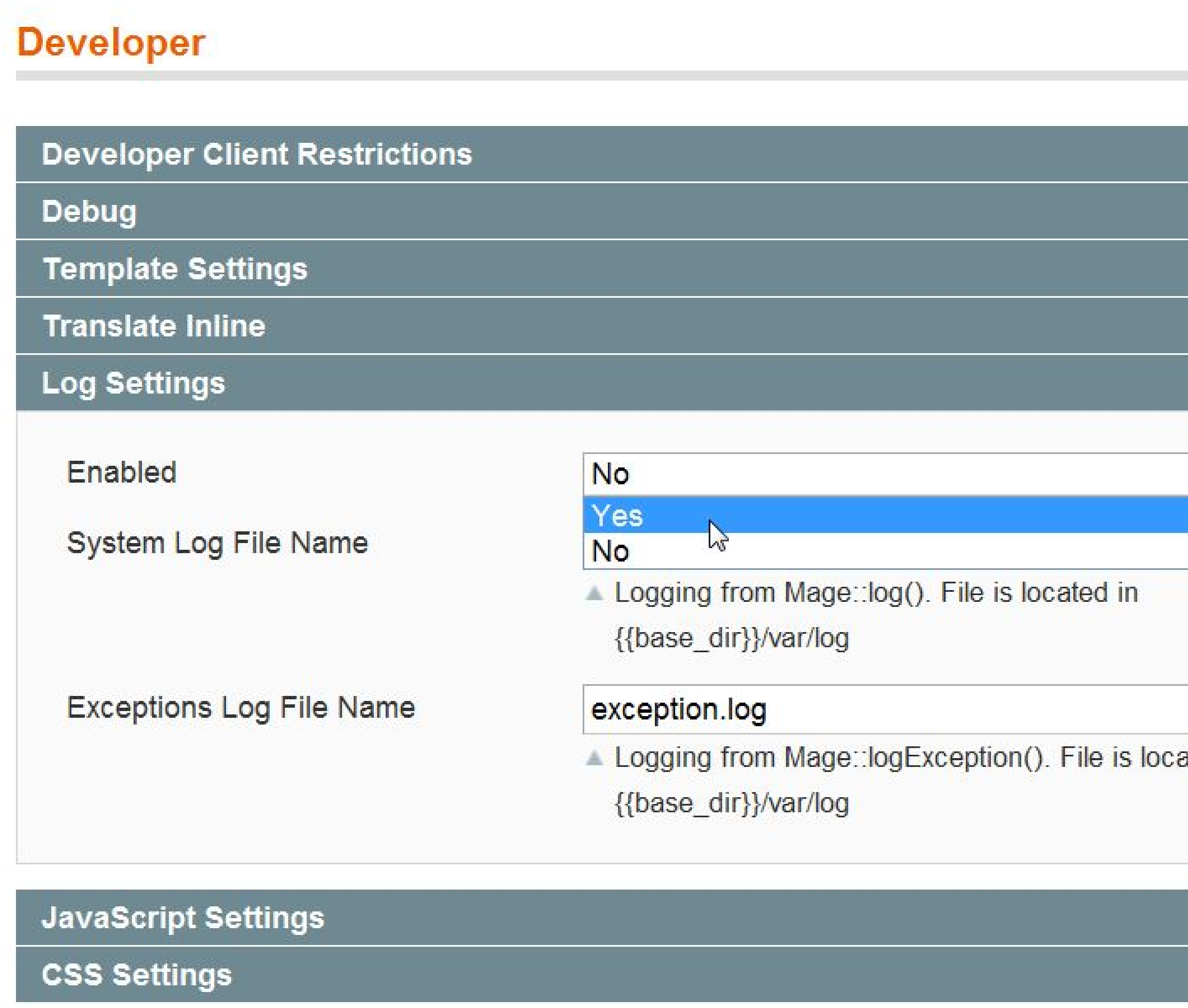
Chapter 20: Magento error handling, messages and reports

Remarks

Error Log Locations

`/var/log/`

Typically the system.log and exception.log file will exist in the `/var/log/` folder. These contain most of the information you will need. You can check to see if these are enabled and what the names of the exception and system log are by going to `System > Configuration > System > Developer > Log Settings`.



Enabled: No (Yes / No)

System Log File Name

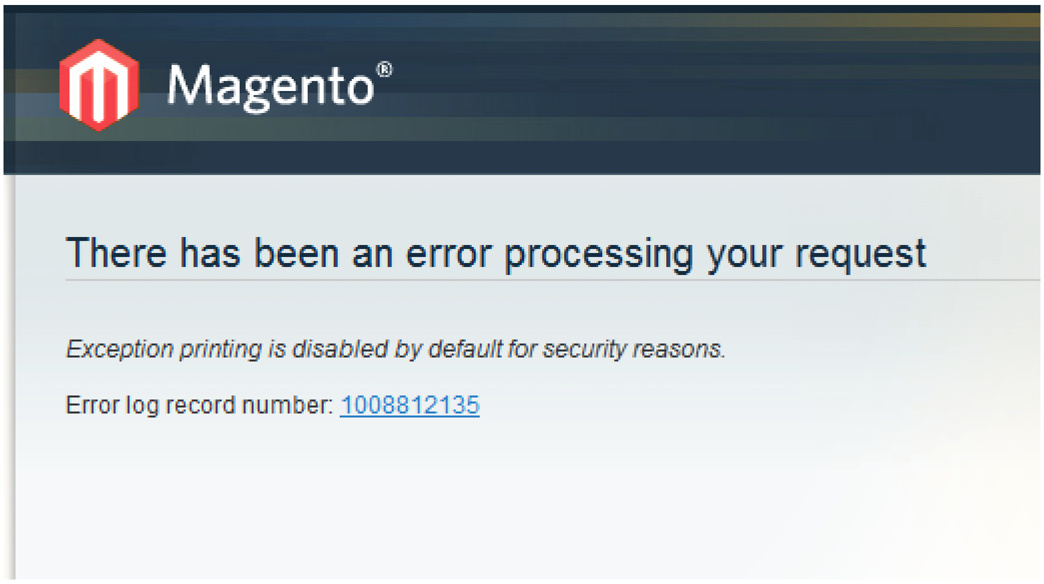
Logging from Mage::log(). File is located in {{base_dir}}/var/log

Exceptions Log File Name: exception.log

Logging from Mage::logException(). File is loca
{{base_dir}}/var/log

`/var/report/`

Report files are generated in this folder after a user has encountered an error. Each file only includes the details for one error. These are used in order to hide the error details from the public. On the error page there will be a report number which is a reference to the corresponding file with the same name in the `/var/report/` folder.



There has been an error processing your request

Exception printing is disabled by default for security reasons.

Error log record number: 1008812135

Examples

Enable displaying of error reporting

In Index page change the following:

```
error_reporting(E_ALL | E_STRICT);
```

to

```
error_reporting(E_ALL);
```

Set `$_SERVER['MAGE_IS_DEVELOPER_MODE'] = true`

and uncomment this line (remove the `#`)

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

You can also Set Dev Mode using `SetEnv` in your `.htaccess` file

To make the error readable and easier to understand, do the following:

1. Open your Magento installation directory. Go to the errors folder.
2. Rename `local.xml.sample` file to `local.xml`.
3. Refresh the error page in browser.

Read Magento error handling, messages and reports online:
https://riptutorial.com/magento/topic/2369/magento-error-handling--messages-and-reports

Chapter 21: Model

Examples

Load model

You can load Magento model using the following code: Mage::getModel('modulename/modelname')

Example: Mage::getModel('catalog/product') This will load Mage_Catalog_Model_product

Create an empty model

To create a new model in your module Add a folder `Model` in your module root folder and create a file Modelname.php in this folder. for example Rick/Demo/Model/Modelname.php

The class name of your model does matter call it like this:

```
<?php
class Rick_Demo_Model_Modelname {

}
```

make sure your model is defined in your `config.xml` in the `etc` folder of your module

Here an example:

2.0.4 Rick_Demo_Model

To load your module use the following code:

```
Mage::getModel('customemodelname/modelname')
```

Read Model online: https://riptutorial.com/magento/topic/7665/model

Chapter 22: Module structure

Remarks

Modules exist to be extended. You cannot change `the app/code/` files without prohibiting any future updates. Instead we add a module to the `app/code/local` directory (the local directory may be missing, if so, it needs to be manually created. This is common in later versions of Magento) for added custom local functionality.

All Module config files begin with a `<config>` tag. The new module is declared inside of the `<modules>` tag. We will be calling a module named YOUR_COMPANY_HelloWorld, therefore the `<YOUR_COMPANY_HelloWorld>` tags are used. In here we define the version (very first = 0.0.1)

A list of **XML events** can be found at: http://www.magentocommerce.com/wiki/5_-_modules_and_development/reference/module_config.xml

If you are having any trouble then check out: http://coding.smashingmagazine.com/2012/11/30/introducing-magento-layout/

Examples

Creating a module from scratch (Hello World)

Magento custom module development is a core part of any Magento development or Magento project, because at any stage you may want to integrate your own functionality/module in your existing Magento project.

The first thing developers should disable is the system cache. Otherwise developing will become a pain as any changes will require a flush. From the magento admin panel: navigate to `System > Cache Management > Select All > Actions : Disable`. Use the following guide to creating a new module:

- Create a folder in `app/code/local/` - naming conventions usually take the name of your company e.g. `app/code/local/<YOUR_COMPANY>`.
- Now we have a custom location for our modules. Create another directory, call it something related to the type of module you want to create e.g. app/code/local/<YOUR_COMPANY>/HelloWorld/ - “HelloWorld” is what I will call this Module.
- This directory needs a `config.xml` so Magento recognises it as a new module. Create another folder named `etc`. Followed by a an xml file called config.xml. The directory should look like `app/code/local/<YOUR_COMPANY/HelloWorld/etc/config.xml` This is the structure of the xml file:

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

```
   <modules>
      <YOUR_COMPANY_HelloWorld>
         <version>0.0.1</version>
      </YOUR_COMPANY_HelloWorld>
   </modules>
</config>
```

- Next, Modules need to be declared to Magento. Proceed to `app/etc/modules`. Create another XML document and give it chosen names of your tags: `YOUR_COMPANY_HelloWorld` in my case. In this document write:

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
   <modules>
      <YOUR_COMPANY_HelloWorld>
         <!-- Whether our module is active: true or false -->
         <active>true</active>
         <!-- Which code pool to use: core, community or local -->
         <codePool>local</codePool>
      </YOUR_COMPANY_HelloWorld>
   </modules>
</config>
```

- Again config and module tags are used to declare a new module to Magento. Active is the default value which can be accessed in the `Admin Panel under System > Configuration > Advanced`. `codePool` tells Magento which directory to look in. `local` in our case
- This module has now been set up, thus the Model of our MVC structure. you should be able to see your new module in Admin Panel under `System > Configuration > Advanced`. **However, it does not do anything yet!** You will need to go back to our config.xml file and define XML elements.
- Following on with the tutorial; We will use some of these XML Elements to create classes and manipulate all pages in the frontend of our site. Back to the `config.xml` file write the following under the `</modules>` tag:

```
<global>
   <!-- adding a new block definition -->
   <blocks>
       <!-- A unique short name for our block files -->
       <helloworld>
          <!-- the location of our modules block -->
          <class>YOUR_COMPANY_HelloWorld_Block</class>
       </helloworld>
   </blocks>
</global>
<!-- We are making changes to the frontend -->
<frontend>
   <!-- We are making changes to the layout of the front end -->
   <layout>
      <!-- we are adding a new update file -->
      <updates>
         <!-- Creating the name of our update and linking it the module -->
         <helloworld module="YOUR_COMPANY_HelloWorld">
```

```
              <!-- The name of the layout file we are adding -->
              <file>helloworld.xml</file>
          </helloworld>
      </updates>
   </layout>
</frontend>
```

- As you can see we are constantly extending rather than manipulating core files. The `helloworld` tag is lower case because that will point to a Handle, and for continuity we will name it as closely as possible. We then link this to the `YOUR_COMPANY_HelloWorld` module.
- We are changing the layout. Therefore we need to create this Handle in the layout directory. proceed to `app/design/frontend/base/default/layout`. We told the module to look for the `helloworld.xml` file. Therefore we must create it in this directory. What are you waiting for. Do it!! and populate it with:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Layout files begin with this code -->
<layout>
   <!-- this is the layout handle. Default is handled on all pages. We want this module
to execute on all pages -->
   <default>
      <!-- This is the block we want to bolt onto the existing before_body_end block -->
      <reference name="before_body_end">
          <!-- We define our new block and template to be added to before_body_end -->
          <block name="helloworld_footer" template="helloworld/footer.phtml"
type="helloworld/footer"/>
      </reference>
   </default>
</layout>
```

- Now those of you who have a little Magento experience, or have read any more noteworthy Magento tutorials, may be gasping at the fact we are making changes in base/default since this is where Magento core files are located. However, we are not modifying any files here, we are creating new ones, and furthermore we are prefixing our file name with “helloworld,” so there is very little chance of this conflicting with other modules or causing any issues with upgrading Magento in the future. Happy days!
- Because we want to affect all pages, we use the default tag and Reference it to the `before_body_end` Structural Block. This will act the role of the Action and trigger the View section of our MVC structure.
- Now we understand that we are bolting onto the `before_body_end` block. and linking it to our custom block. This is called a Reference and is a **hook**. We currently are not hooking it to anything existing therefore we must create the necessary files.
- In `helloworld.xml` we stated in template a `footer.phtml`. Proceed to `app/design/frontend/base/default/template` and create a directory `helloworld`.
- Inside this directory create the `footer.phtml` file and fill in with HTML, this tutorial simply writes this to display some PHP functionality linked with our PHTML file:

```
<p>Hello Everyone! Todays date is <?php echo $this->getDate() ?></p>
```

- We now need to create our own block object to couple the template with our block functionality. Create the directory app/code/local/YOUR_COMPANY/HelloWorld/Block/ and create the file `Footer.php` inside of this. This was referenced in our zeta_layout.xml in the type “helloworld/footer”. Populate this file with:

```
<?php
class YOUR_COMPANY_HelloWorld_Block_Footer extends Mage_Core_Block_Template {
   public function getDate() {
      $date = date('Y-m-d');
      return urlencode($date);
   }
}
?>
```

- This is the functionality that will populate our call `getDate()` we called from our `.phtml` file. We extend the `Mage_Core_Block_Template`.
- This functionality is now complete. Test this out by going to your home page where you should see your module within the footer of every page!

Read Module structure online: https://riptutorial.com/magento/topic/3962/module-structure

Chapter 23: MVC Structure

Remarks

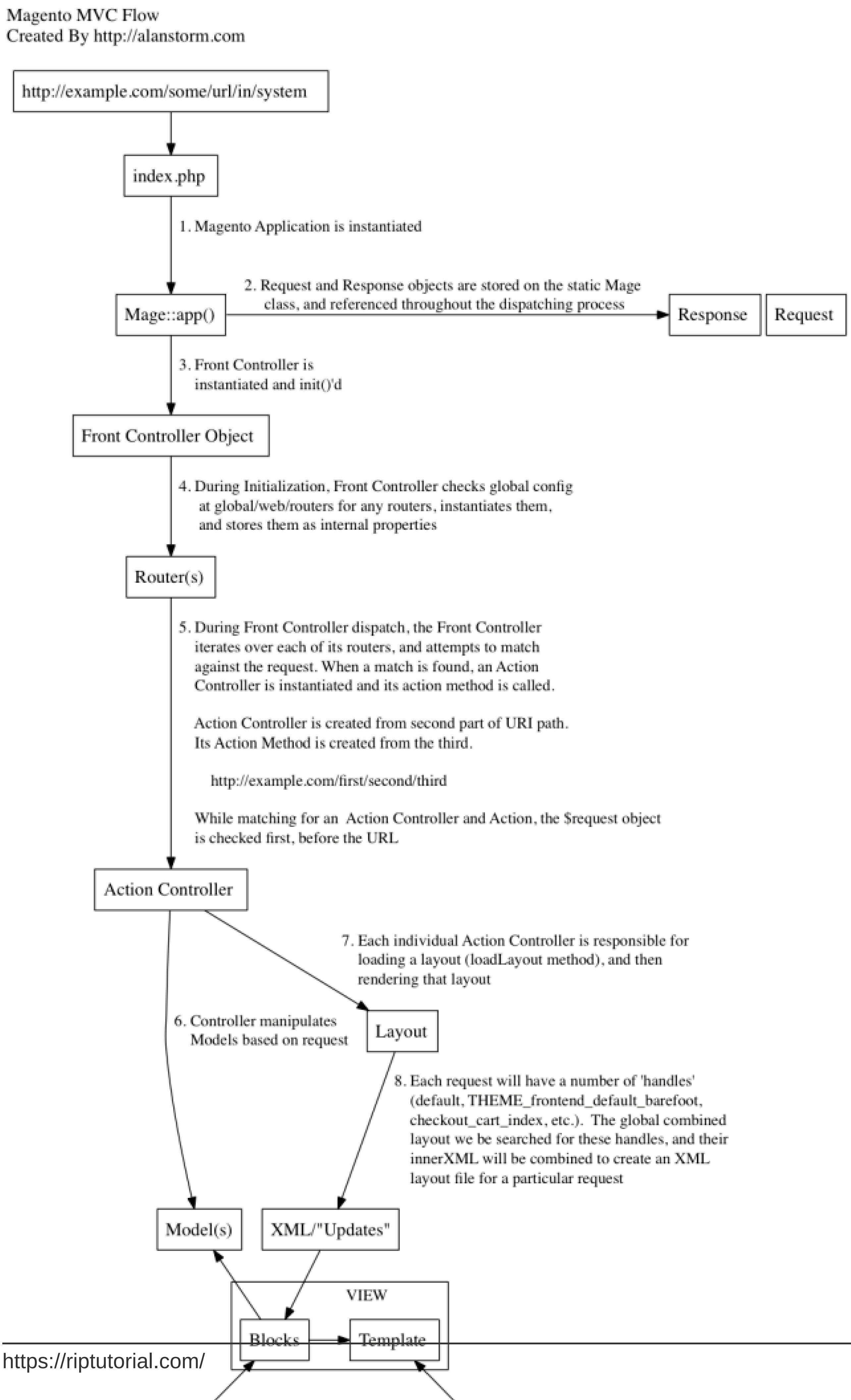
MVC stands for Model-View-Controller. Any application that separates it’s data access, business logicand user interface is called MVC. There can be two types of MVC: convention-based and configuration-based. Example, cakePHP is convention-based, i.e. you just need to follow the instructions of the core system to get your module ready in just few lines. Magento is configuration-based, i.e. you need to specify each and every thing to your module’s config file in order to get it work. Magento has Controller (for Routing), Block, Model and Template file. How Magento’s MVC works:

1. When you enter the URL (something like http://mysite.com/frontname/controller/method/param1/value1/param2/value2), this URL is intercepted by one PHP file called index.php which instantiates Magento application
2. Magento application instantiates Front Controller object
3. Further, front controller instantiates Router objects (specified in module’s config.xml, global tag)
4. Now, Router is responsible to “match” the frontname which is in our URL
5. If “match” is found, it sees controller name and method name in the URL, which is finally called.
6. Now depending on what is written in action name (method name), it is executed. If any models are called in it, the controller method will instantiate that model and call the method in it which is requested.
7. Then the controller action (method) instantiate the Layout object, which calls Block specified for this action (method) name (Each controller action name have block and template file associated with it, which can be found at app/design/frontend or adminhtml/namespace/module/layout/module.xml file, name of layout file (module.xml) can be found in config.xml of that module, in layout updates tag).
8. Template file (.phtml) now calls the corresponding block for any method request. So, if you write $this->methodName in .phtml file, it will check “methodName” in the block file which is associated in module.xml file.
9. Block contains PHP logic. It references Models for any data from DB.
10. If either Block, Template file or Controller need to get/set some data from/to database, they can call Model directly like Mage::getModel(‘modulename/modelname’).

Examples

Understand MVC in Magento

MVC Flow in Magento


Magento MVC Flow
Created By http://alanstorm.com
http://example.com/some/url/in/system
index.php
1. Magento Application is instantiated
Mage::app()
2. Request and Response objects are stored on the static Mage class, and referenced throughout the dispatching process
Response
Request
3. Front Controller is instantiated and init()'d
Front Controller Object
4. During Initialization, Front Controller checks global config at global/web/routers for any routers, instantiates them, and stores them as internal properties
Router(s)
5. During Front Controller dispatch, the Front Controller iterates over each of its routers, and attempts to match against the request. When a match is found, an Action Controller is instantiated and its action method is called.
Action Controller is created from second part of URI path. Its Action Method is created from the third.
http://example.com/first/second/third
While matching for an Action Controller and Action, the $request object is checked first, before the URL
Action Controller
7. Each individual Action Controller is responsible for loading a layout (loadLayout method), and then rendering that layout
6. Controller manipulates Models based on request
Layout
8. Each request will have a number of 'handles' (default, THEME_frontend_default_barefoot, checkout_cart_index, etc.). The global combined layout we be searched for these handles, and their innerXML will be combined to create an XML layout file for a particular request
Model(s)
XML/"Updates"
VIEW
Blocks
Template

Chapter 24: Optimizing Magento For Speed

Examples

Optimizing Magento Changing The .htaccess File

Magento is a very popular eCommerce application. It offers a great deal of customization and abilities from initial install. Here are a few suggestions for optimizing a Magento installation.

Enabling Output Compression

In your .htaccess file for Magento you will find a section of text starting with the line,

```
<IfModule mod_deflate.c> and ending at </IfModule>
```

This section of code can be used to turn on Apache's mod_deflate module, which provides compression for text, css, and javascript. You will want to uncomment (remove the # symbol) multiple lines so that it looks like this:

###

enable apache served files compression

http://developer.yahoo.com/performance/rules.html#gzip

```
# Insert filter on all content
SetOutputFilter DEFLATE
# Insert filter on selected content types only
AddOutputFilterByType DEFLATE text/html text/plain text/xml text/css text/javascript

# Netscape 4.x has some problems...
BrowserMatch ^Mozilla/4 gzip-only-text/html

# Netscape 4.06-4.08 have some more problems
BrowserMatch ^Mozilla/4\.0[678] no-gzip

# MSIE masquerades as Netscape, but it is fine
BrowserMatch \bMSIE !no-gzip !gzip-only-text/html

# Don't compress images
SetEnvIfNoCase Request_URI \.(?:gif|jpe?g|png)$ no-gzip dont-vary

# Make sure proxies don't deliver the wrong content
Header append Vary User-Agent env=!dont-vary
</IfModule>
```

Enabling Expires Headers

First-time visitors to any web page has to make several HTTP requests. By using the "Expires" header you make the components of the requests cacheable. This avoids unnecessary HTTP requests on subsequent page views.

You want to find the area of the .htaccess file that starts with <IfModulemod_expires.c> and ends with the first you see after it, and make it look like this:

```
<IfModule mod_expires.c>

############################################
## Add default Expires header
## http://developer.yahoo.com/performance/rules.html#expires
    ExpiresActive On
    ExpiresDefault "access plus 1 year"

</IfModule>
```

Admin Settings

Merge JS and CSS Files

This particular tweak will reduce the number of HTTP requests on your eCommerce site. [box type=”alert” border=”full”]Note: This can sometimes break some applications. After performing the following steps, please ensure that the site still performs as it did before enabling this feature.[/box]

1. Login to your administration area and go to – System > Configuration > Developer
2. Under “JavaScript Settings”, change “Merge JavaScript Files” to yes.
3. Under “CSS Settings”, change “Merge CSS Files” to yes.
4. Finally you will want to clear your Magento cache.

Enable Flat Catalogs

The model Magento uses to store customer and product data results in longer than average SQL queries and more reads. Enabling the Flat Catalog option for Categories and Products will merge product data into one table, therefore improving performance.

Login to your administration area and go to – System > Configuration > Catalog Under “Frontend”, change “Use Flat Catalog Category” to yes. Under “Frontend”, change “Use Flat Catalog Product” to yes – this is optional. Next, you will want to clear your Magento cache. Finally, you will need to reindex the tables. Enable Compilation

[box type=”alert” border=”full”]Note: This can sometimes break some applications. After performing the following steps, please ensure that the site still performs as it did before enabling this feature.[/box]

1. Login to your administration area and go to – System > Tools > Compilation

2. Next, simply click the Run Compilation Process Button

3. After the compilation has run, it should enable itself automatically

Enable System Cache

1. Login to your administration area and go to – System > Cache Management
2. Next, click on the Select All link
3. Finally, make sure the Actions is set to Enable and click submit

Disable Error Logging

Login to your administration area and go to – System > Configuration > Developer Under the Log Settings section, be sure that Enabled is set to No Database Maintenance Tips

There are several tables used by Magento for logging. While logging is very important regarding knowing what has and is going on with your store, the logs can become large very quickly, so regularly maintenance can be of great assistance.

Here are the tables for logging:

```
log_customer
log_visitor
log_visitor_info
log_url
log_url_info
log_quote
report_viewed_product_index
report_compared_product_index
report_event
catalog_compare_item
```

Read Optimizing Magento For Speed online: https://riptutorial.com/magento/topic/8010/optimizing-magento-for-speed

Chapter 25: Orders

Examples

Get order by ID

```
$orderid = 12345;
$order = Mage::getModel('sales/order')->load($orderid);
```

The above code is roughly analogous to the following SQL query.

```
select * from sales_flat_order where entity_id=12345;
```

Get order by Increment ID

```
$incrementid = 100000000;
$order = Mage::getModel('sales/order')->loadByIncrementId($incrementid);
```

The above code is roughly analogous to the following SQL query.

```
select * from sales_flat_order where increment_id=100000000;
```

The `increment_id` is the customer facing order identifier, whereas the `entity_id` is the database level identifier for the order.

Add comment to order history

You can add comment and status to order. Get order :

```
$orderid = 12345;
$order = Mage::getModel('sales/order')->load($orderid);
```

And add comment:

```
//$isNotify means you want to notify customer or not.

$order->addStatusToHistory($status, $message, $isNotify);
$order->save()
```

Read Orders online: https://riptutorial.com/magento/topic/1556/orders

Chapter 26: Product Image Urls

Introduction

Get product image urls for thumbnail, small image and base image. Get cached as well and non caches direct media urls.

Examples

Cached Image urls

```
Mage::helper('catalog/image')->init($item->getProduct(), 'thumbnail');
Mage::helper('catalog/image')->init($item->getProduct(), 'small_image');
Mage::helper('catalog/image')->init($item->getProduct(), 'image');
```

Non cached Image Urls from Media

```
Mage::getModel('catalog/product_media_config')->getMediaUrl($product->getThumbnail());
//Thumbnail
Mage::getModel('catalog/product_media_config')->getMediaUrl($product->getSmallImage());
//Small Image
Mage::getModel('catalog/product_media_config')->getMediaUrl($product->getImage()); //Base
```

Read Product Image Urls online: https://riptutorial.com/magento/topic/9262/product-image-urls

Chapter 27: Quick Task Cheat Sheet

Remarks

https://gist.github.com/arosenhagen/2397824

Many many snippets very helpful

Examples

Get product Stock Qty

load($id); // or load it by SKU // $sku = "microsoftnatural"; // $_product = Mage::getModel('catalog/product')->loadByAttribute('sku', $sku); $stock = Mage::getModel('cataloginventory/stock_item')->loadByProduct($_product); print_r($stock->getData()); echo $stock->getQty(); echo $stock->getMinQty(); echo $stock->getMinSaleQty();

Read Quick Task Cheat Sheet online: https://riptutorial.com/magento/topic/7060/quick-task-cheat-sheet

Chapter 28: Rendering

Remarks

Customising Core Functionality with Themes

Themes have layout files which, amongst other things, can be used to change which blocks appear on the page. The block template can also be changed and different methods called.

Store-level designs

Magento's hierarchical structured themes means that a base theme can be extended and assigned on a store level.

Registering Custom Themes

Themes can be configured in three ways:

1. Per store under System > Configuration > Design.
2. Design change with time limits System > Design.
3. Theme exceptions can also be set on a category and product level.

Package versus Theme

A package has multiple themes. Each of the themes in a package inherits from the default theme within a package.

Design Fallback

The theme fallback procedure for locating templates files is:

1. {package}/{theme}
2. {package}/default
3. base/default

To add further directories to the theme fallback mechanism Mage_Core_Model_Design_Package::getFilename method need to be rewritten

For the admin area the fallback is default/default.

Template and Layout Paths

Blocks

Blocks are used for output. The root block is the parent of all blocks and is of type Mage_Page_Block_Html.

Mage_Core_Block_Template blocks use template files to render content. The template file name are set within setTemplate() or addData('template') with relative paths.

Templates are just pieces of PHP included in Mage_Core_Block_Template. Therefore $this in a template refers to the block.

Mage_Core_Block_Template uses a buffer before including a template to prevent premature output.

The Mage_Core_Model_Layout::createBlock method creates instances of blocks.

The Mage_Core_Model_Layout_Update class considers which blocks need to be created for each page by looking at the layout handles.

All output blocks are rendered, e.g. by calling toHtml(), which in turn can choose to render their children.

Text and Text_List blocks automatically render their content.

There are two events that are fired around block rendering that can be used to modify the block before and after rendering the HTML:

core_block_abstract_to_html_before core_block_abstract_to_html_after A child block will only be rendered automatically if it is of class Mage_Core_Block_Textlist otherwise the getChildHtml method needs to be called.

Block instances can be accessed through the layout, e.g. Mage::app()->getLayout() and $controller->getLayout(). Block output is controlled by the _toHtml() function.

Templates are rendered by the renderView()/fetchView() methods inside a template block. Output buffering can be disabled with $layout->setDirectOutput.

It is possible to add a block to the current layout but it needs to be done before the renderLayout() method is called.

Layout XML

```
<reference>
  -edit a block
<block>
  - define a block
<action>
  - call method on a block
<update>
  - include nodes from another handle.
```

Layout files can be registered in config.xml:

```
<config>
    <{area}>
        <layout>
            <updates>
                <{name}>
                    <file>{filepath}</file>
                </{name}>
            </updates>
        </layout>
    </{area}>
</config>
```

Page output can be customised in the following ways:

- Template changes
- Layout changes
- Overriding blocks
- Observers Variables on blocks can be set in the following ways:
- Layout -Through actions or attributes
- Controller -$this-getLayout()->getBlock()
- Child blocks -$this->getChild()
- Other -Mage::app()->getLayout()

Head Block Assets

JavaScript and CSS assets are handled in the Mage_Page_Block_Html_head block. This block handles the merging of assets into a single file to minimise HTTP requests. The merged file is based on the edit time of the source files.

When merging CSS, a callback function on Mage_Core_Model_Design_Package is called to update any @import or url() directives with the correct URLs

Examples

Different mechanisms for disabling block output

- If the response has already been created and set on the response object outside the regular rendering process (e.g., in an observer), the 'no-renderLayout' flag can be set on the action controller using

```
Mage::app()->getFrontController()->getAction()->setFlag('','no-renderLayout');
```

- This prevents `renderLayout()` from processing the output blocks.
- The same can be achieved by calling setNoRender(true) on the front controller: `Mage::app()->getFrontController()->setNoRender(true);`
- Setting the `isDispatched()` flag on the response object might be more efficient to achieve a similar effect.

different types of blocks

- Mage_Core_Block_Template
- Mage_Core_Block_Text_List
- Mage_Core_Block_Messages
- Mage_Core_Block_Text_Tag
- Mage_Core_Block_Text

- Mage_Page_Block_Template_Links All blocks are either **structural blocks** or All blocks are either structural blocks or content blocks. Example:
- **core/text_list**

Example of a **structural block**.

It does not utilize templates; it's simply used to output the content of all of its child blocks one after the other.

- **core/template**

Example of a **content block**.

The output of this type of block depends on the assigned template. It's child blocks are output within it's template via the getChildHtml('block_name') method.. Example: core/text_list - Example of a structural block. It does not utilize templates; it's simply used to output the content of all of its child blocks one after the other. core/template - Example of a content block. The output of this type of block depends on the assigned template. It's child blocks are output within it's template via the getChildHtml('block_name') method.

Block instances can be accessed from the controller

From an action controller:

```
$this->getLayout()->getBlock('head')->getTemplate();

/**
* Get specified tab grid
*/
public function gridOnlyAction()
{
$this->_initProduct();
$this->getResponse()->setBody(
$this->getLayout()->createBlock('adminhtml/catalog_product_edit_tab_' .
$this->getRequest()->gerParam('gridOnlyBlock')
)
->toHtml()
);
}
```

Read Rendering online: https://riptutorial.com/magento/topic/7140/rendering

Chapter 29: Shell, CLI

Remarks

Basics

- You need to have a Linux command line or connect using SSH to your server in order to use shell scripts.
- Go to your `MAGENTO_ROOT/shell`
- Script can be run by typing i.e.

```
php -f indexer.php help
```

Core shell methods by files

1. abstract.php
2. indexer.php
3. compiler.php
4. log.php

Custom php shell scripts

Sometimes we need to access Magento outside of a webbrowser to ommmit execution times or set different things that won't affect the frontend.

There are 2 ways to bootstrap Magento but only one is the Magento way. Read more above in examples section.

Examples

Using shell without extending Mage_Shell_Abstract

Bootstrapping Magento by calling:

```
require_once 'app/Mage.php';
Mage::app();
// Your code
```

This is the simplest way but not really the Magento way because we're not using class that extends `Mage_Shell_Abstract` - the class which when extended provides us with tools to parse command line arguments, calls `__applyPhpVariables()` in it's constructor (function parses .htaccess files and applies php settings to shell script).

Using shell the Magento way - extend Mage_Shell_Abstract

Magento way

File resides in `shell/custom.php`

```
<?php
require_once' abstract.php';

class Stackoverflow_Shell_Custom extends Mage_Shell_Abstract
{

protected $_argname = array();

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

        // Time limit to infinity
        set_time_limit(0);

        // Get command line argument named "argname"
        // Accepts multiple values (comma separated)
        if($this->getArg('argname')) {
            $this->_argname = array_merge(
                $this->_argname,
                array_map(
                    'trim',
                    explode(',', $this->getArg('argname'))
                )
            );
        }
    }

 // Shell script point of entry
    public function run() {

    }

    // Usage help
    public function usageHelp()
    {
        return <<<USAGE
Usage:  php -f scriptname.php -- [options]

  --argname <argvalue>       Argument description

  help                   This help

USAGE;
    }
}
// Instantiate
```

```
$shell = new Stackoverflow_Shell_Custom();

// Initiate script
$shell->run();

}
```

Performing Reindex from CLI

View Status:

```
php indexer.php status
```

Reindex All

```
php indexer.php reindexall
```

Reindex Specific Index

```
php indexer.php --reindex CODE (see list below)
```

List of Individual Codes

Index	Code
Product Attributes	catalog_product_attribute
Product Prices	catalog_product_price
Catalog URL Rewrites	catalog_url
Product Flat Data	catalog_product_flat
Category Flat Data	catalog_category_flat
Category Products	catalog_category_product
Catalog Search Index	catalogsearch_fulltext
Stock Status	cataloginventory_stock

Read Shell, CLI online: https://riptutorial.com/magento/topic/5387/shell--cli

Chapter 30: Specific urls

Examples

Cart url

```
$this->helper('checkout/url')->getCartUrl();
```

OR

```
Mage::helper('checkout/url')->getCartUrl();
```

Checkout url

```
$this->helper('checkout/url')->getCheckoutUrl();
```

OR

```
Mage::helper('checkout/url')->getCheckoutUrl();
```

Login url

```
$this->helper('customer/data')->getLoginUrl();
```

OR

Mage::helper('customer/data')->getLoginUrl();

Logout url

```
$this->helper('customer/data')->getLogoutUrl();
```

OR

```
Mage::helper('customer/data')->getLogoutUrl();
```

Forgot password url

```
$this->helper('customer/data')->getForgotPasswordUrl();
```

OR

```
Mage::helper('customer/data')->getForgotPasswordUrl();
```

Account customer url

```
$this->helper('customer/data')->getAccountUrl();
```

OR

```
Mage::helper('customer/data')->getAccountUrl();
```

Media, JS, Skin URL

To Retrieve URL path in STATIC BLOCK Or CMS pages

To get SKIN URL

```
{{skin url='images/sampleimage.jpg'}}
```

To get Media URL

```
{{media url='/sampleimage.jpg'}}
```

To get Store URL

```
{{store url='mypage.html'}}
```

To get Base URL

```
{{base url=''}}
```

TO Retrieve URL path in PHTML

Not secure Skin URL

```
<?php echo $this->getSkinUrl('images/sampleimage.jpg') ?>
```

Secure Skin URL

```
<?php echo $this->getSkinUrl('images/ sampleimage.gif',array('_secure'=>true)) ?>
```

Get Current URL

```
<?php $current_url = Mage::helper('core/url')->getCurrentUrl();?>
```

Get Home URL

```
<?php $home_url = Mage::helper('core/url')->getHomeUrl();?>
```

Get Magento Media Url

```
<?php Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_LINK);?>
<?php Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_MEDIA);?>
```

Get Magento Skin Url

```
<?php Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_SKIN);?>
```

Get Magento Store Url

```
<?php Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_WEB);?>
```

Get Magento Js Url

```
<?php Mage::getBaseUrl(Mage_Core_Model_Store::URL_TYPE_JS);?>
```

Read Specific urls online: https://riptutorial.com/magento/topic/2144/specific-urls

Chapter 31: Sql script to delete test data

Introduction

Sql script to delete test data of products, customers, logs and sales.

Examples

Delete Customer test data

```
SET FOREIGN_KEY_CHECKS=0;

-- Customers
TRUNCATE `customer_address_entity`;
TRUNCATE `customer_address_entity_datetime`;
TRUNCATE `customer_address_entity_decimal`;
TRUNCATE `customer_address_entity_int`;
TRUNCATE `customer_address_entity_text`;
TRUNCATE `customer_address_entity_varchar`;
TRUNCATE `customer_entity`;
TRUNCATE `customer_entity_datetime`;
TRUNCATE `customer_entity_decimal`;
TRUNCATE `customer_entity_int`;
TRUNCATE `customer_entity_text`;
TRUNCATE `customer_entity_varchar`;
ALTER TABLE `customer_address_entity` AUTO_INCREMENT=1;
ALTER TABLE `customer_address_entity_datetime` AUTO_INCREMENT=1;
ALTER TABLE `customer_address_entity_decimal` AUTO_INCREMENT=1;
ALTER TABLE `customer_address_entity_int` AUTO_INCREMENT=1;
ALTER TABLE `customer_address_entity_text` AUTO_INCREMENT=1;
ALTER TABLE `customer_address_entity_varchar` AUTO_INCREMENT=1;
ALTER TABLE `customer_entity` AUTO_INCREMENT=1;
ALTER TABLE `customer_entity_datetime` AUTO_INCREMENT=1;
ALTER TABLE `customer_entity_decimal` AUTO_INCREMENT=1;
ALTER TABLE `customer_entity_int` AUTO_INCREMENT=1;
ALTER TABLE `customer_entity_text` AUTO_INCREMENT=1;
ALTER TABLE `customer_entity_varchar` AUTO_INCREMENT=1;

-- Search
TRUNCATE `catalogsearch_query`;
TRUNCATE `catalogsearch_fulltext`;
TRUNCATE `catalogsearch_result`;
ALTER TABLE `catalogsearch_query` AUTO_INCREMENT=1;
ALTER TABLE `catalogsearch_fulltext` AUTO_INCREMENT=1;
ALTER TABLE `catalogsearch_result` AUTO_INCREMENT=1;

-- Polls
TRUNCATE `poll`;
TRUNCATE `poll_answer`;
TRUNCATE `poll_store`;
TRUNCATE `poll_vote`;
ALTER TABLE `poll` AUTO_INCREMENT=1;
ALTER TABLE `poll_answer` AUTO_INCREMENT=1;
ALTER TABLE `poll_store` AUTO_INCREMENT=1;
ALTER TABLE `poll_vote` AUTO_INCREMENT=1;
```

```
-- Reports
TRUNCATE `report_viewed_product_index`;
ALTER TABLE `report_viewed_product_index` AUTO_INCREMENT=1;

-- Newsletter
TRUNCATE `newsletter_queue`;
TRUNCATE `newsletter_queue_link`;
TRUNCATE `newsletter_subscriber`;
TRUNCATE `newsletter_problem`;
TRUNCATE `newsletter_queue_store_link`;
ALTER TABLE `newsletter_queue` AUTO_INCREMENT=1;
ALTER TABLE `newsletter_subscriber` AUTO_INCREMENT=1;
ALTER TABLE `newsletter_problem` AUTO_INCREMENT=1;
ALTER TABLE `newsletter_queue_store_link` AUTO_INCREMENT=1;

-- Wishlist
TRUNCATE `wishlist`;
ALTER TABLE `wishlist` AUTO_INCREMENT=1;

SET FOREIGN_KEY_CHECKS=1;
```

Delete Product test data

```
SET FOREIGN_KEY_CHECKS = 0;

TRUNCATE TABLE `catalog_product_bundle_option`;
TRUNCATE TABLE `catalog_product_bundle_option_value`;
TRUNCATE TABLE `catalog_product_bundle_selection`;
TRUNCATE TABLE `catalog_product_entity_datetime`;
TRUNCATE TABLE `catalog_product_entity_decimal`;
TRUNCATE TABLE `catalog_product_entity_gallery`;
TRUNCATE TABLE `catalog_product_entity_int`;
TRUNCATE TABLE `catalog_product_entity_media_gallery`;
TRUNCATE TABLE `catalog_product_entity_media_gallery_value`;
TRUNCATE TABLE `catalog_product_entity_text`;
TRUNCATE TABLE `catalog_product_entity_tier_price`;
TRUNCATE TABLE `catalog_product_entity_varchar`;
TRUNCATE TABLE `catalog_product_link`;
TRUNCATE TABLE `catalog_product_link_attribute`;
TRUNCATE TABLE `catalog_product_link_attribute_decimal`;
TRUNCATE TABLE `catalog_product_link_attribute_int`;
TRUNCATE TABLE `catalog_product_link_attribute_varchar`;
TRUNCATE TABLE `catalog_product_link_type`;
TRUNCATE TABLE `catalog_product_option`;
TRUNCATE TABLE `catalog_product_option_price`;
TRUNCATE TABLE `catalog_product_option_title`;
TRUNCATE TABLE `catalog_product_option_type_price`;
TRUNCATE TABLE `catalog_product_option_type_title`;
TRUNCATE TABLE `catalog_product_option_type_value`;
TRUNCATE TABLE `catalog_product_super_attribute`;
TRUNCATE TABLE `catalog_product_super_attribute_label`;
TRUNCATE TABLE `catalog_product_super_attribute_pricing`;
TRUNCATE TABLE `catalog_product_super_link`;
TRUNCATE TABLE `catalog_product_enabled_index`;
TRUNCATE TABLE `catalog_product_website`;
TRUNCATE TABLE `catalog_product_entity`;
TRUNCATE TABLE `cataloginventory_stock_item`;
TRUNCATE TABLE `cataloginventory_stock_status`;
TRUNCATE TABLE `catalog_category_entity`;
TRUNCATE TABLE `catalog_category_entity_datetime`;
TRUNCATE TABLE `catalog_category_entity_decimal`;
TRUNCATE TABLE `catalog_category_entity_int`;
TRUNCATE TABLE `catalog_category_entity_text`;
TRUNCATE TABLE `catalog_category_entity_varchar`;
TRUNCATE TABLE `catalog_category_product`;
TRUNCATE TABLE `catalog_category_product_index`;
TRUNCATE TABLE `catalog_product_relation`;
TRUNCATE TABLE `catalog_product_flat_1`;
TRUNCATE TABLE `catalog_category_flat_store_1`;
TRUNCATE TABLE `catalog_category_flat_store_2`;
TRUNCATE TABLE `catalog_category_flat_store_3`;

-- Tags
TRUNCATE `tag`;
TRUNCATE `tag_relation`;
TRUNCATE `tag_summary`;
ALTER TABLE `tag` AUTO_INCREMENT=1;
ALTER TABLE `tag_relation` AUTO_INCREMENT=1;
ALTER TABLE `tag_summary` AUTO_INCREMENT=1;

SET FOREIGN_KEY_CHECKS = 1;
```

Delete Sales test data

```
SET FOREIGN_KEY_CHECKS=0;

TRUNCATE `sales_payment_transaction`;
TRUNCATE `sales_flat_creditmemo`;
TRUNCATE `sales_flat_creditmemo_comment`;
TRUNCATE `sales_flat_creditmemo_grid`;
TRUNCATE `sales_flat_creditmemo_item`;
TRUNCATE `sales_flat_order`;
TRUNCATE `sales_flat_order_address`;
TRUNCATE `sales_flat_order_grid`;
TRUNCATE `sales_flat_order_item`;
TRUNCATE `sales_flat_order_status_history`;
TRUNCATE `sales_flat_quote`;
TRUNCATE `sales_flat_quote_address`;
TRUNCATE `sales_flat_quote_address_item`;
TRUNCATE `sales_flat_quote_item`;
TRUNCATE `sales_flat_quote_item_option`;
TRUNCATE `sales_flat_order_payment`;
TRUNCATE `sales_flat_quote_payment`;
TRUNCATE `sales_flat_quote_shipping_rate`;
TRUNCATE `sales_flat_shipment`;
TRUNCATE `sales_flat_shipment_item`;
TRUNCATE `sales_flat_shipment_grid`;
TRUNCATE `sales_flat_shipment_track`;
TRUNCATE `sales_flat_shipment_comment`;
TRUNCATE `sales_flat_invoice`;
TRUNCATE `sales_flat_invoice_grid`;
TRUNCATE `sales_flat_invoice_item`;
TRUNCATE `sales_flat_invoice_comment`;
TRUNCATE `sales_order_tax`;
TRUNCATE `sales_order_tax_item`;

-- Reports
```

```
TRUNCATE `sales_bestsellers_aggregated_daily`;
TRUNCATE `sales_bestsellers_aggregated_monthly`;
TRUNCATE `sales_bestsellers_aggregated_yearly`;
TRUNCATE `sales_invoiced_aggregated`;
TRUNCATE `sales_invoiced_aggregated_order`;
TRUNCATE `sales_order_aggregated_created`;
TRUNCATE `sales_order_aggregated_updated`;
TRUNCATE `sales_refunded_aggregated`;
TRUNCATE `sales_refunded_aggregated_order`;
TRUNCATE `sales_shipping_aggregated`;
TRUNCATE `sales_shipping_aggregated_order`;
TRUNCATE `coupon_aggregated`;
TRUNCATE `review`;
TRUNCATE `review_detail`;
TRUNCATE `review_entity_summary`;
TRUNCATE `rating_store`;


ALTER TABLE `sales_payment_transaction` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_creditmemo` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_creditmemo_comment` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_creditmemo_grid` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_creditmemo_item` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_order` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_order_address` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_order_grid` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_order_item` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_order_status_history` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_quote` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_quote_address` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_quote_address_item` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_quote_item` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_quote_item_option` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_order_payment` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_quote_payment` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_quote_shipping_rate` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_shipment` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_shipment_item` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_shipment_track` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_shipment_comment` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_invoice` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_invoice_grid` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_invoice_item` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_invoice_comment` AUTO_INCREMENT=1;
ALTER TABLE `sales_flat_shipment_grid` AUTO_INCREMENT=1;
ALTER TABLE `sales_order_tax` AUTO_INCREMENT=1;
ALTER TABLE `sales_order_tax_item` AUTO_INCREMENT=1;
ALTER TABLE `sales_invoiced_aggregated` AUTO_INCREMENT=1;
ALTER TABLE `sales_invoiced_aggregated_order` AUTO_INCREMENT=1;

TRUNCATE `eav_entity_store`;
ALTER TABLE `eav_entity_store` AUTO_INCREMENT=1;

SET FOREIGN_KEY_CHECKS=1;
```

Delete Logs test data

```
SET FOREIGN_KEY_CHECKS=0;
```

```
TRUNCATE `log_customer`;
TRUNCATE `log_visitor`;
TRUNCATE `log_visitor_info`;
TRUNCATE `log_visitor_online`;
TRUNCATE `log_quote`;
TRUNCATE `log_summary`;
TRUNCATE `log_summary_type`;
TRUNCATE `log_url`;
TRUNCATE `log_url_info`;
TRUNCATE `sendfriend_log`;
TRUNCATE `report_event`;
TRUNCATE `dataflow_batch_import`;
TRUNCATE `dataflow_batch_export`;
TRUNCATE `index_process_event`;
TRUNCATE `index_event`;
ALTER TABLE `log_customer` AUTO_INCREMENT=1;
ALTER TABLE `log_visitor` AUTO_INCREMENT=1;
ALTER TABLE `log_visitor_info` AUTO_INCREMENT=1;
ALTER TABLE `log_visitor_online` AUTO_INCREMENT=1;
ALTER TABLE `log_quote` AUTO_INCREMENT=1;
ALTER TABLE `log_summary` AUTO_INCREMENT=1;
ALTER TABLE `log_url_info` AUTO_INCREMENT=1;
ALTER TABLE `sendfriend_log` AUTO_INCREMENT=1;
ALTER TABLE `report_event` AUTO_INCREMENT=1;
ALTER TABLE `dataflow_batch_import` AUTO_INCREMENT=1;
ALTER TABLE `dataflow_batch_export` AUTO_INCREMENT=1;
ALTER TABLE `index_event` AUTO_INCREMENT=1;

SET FOREIGN_KEY_CHECKS=1;
```

Read Sql script to delete test data online: https://riptutorial.com/magento/topic/9263/sql-script-to-delete-test-data

Chapter 32: Store and Website Data

Introduction

Get magento store and website related data

Examples

Get current store data

```
$store = Mage::app()->getStore();
$storeId = Mage::app()->getStore()->getStoreId();
$storeCode = Mage::app()->getStore()->getCode();
$websiteId = Mage::app()->getStore()->getWebsiteId();
$storeGroupId = Mage::app()->getStore()->getGroupId();
$storeName = Mage::app()->getStore()->getName();
$storeSortOrder = Mage::app()->getStore()->getSortOrder();
$storeIsActive = Mage::app()->getStore()->getIsActive();
```

Read Store and Website Data online: https://riptutorial.com/magento/topic/9266/store-and-website-data

Chapter 33: Understanding product types

Remarks

There are six different product types built-in to Magento.

- Simple

A single stock unit

- Configurable

First of the composite products. Allow customers to configure their product and add a single simple product to basket.

- Grouped

The second composite product, a grouped product relates simple products and provides customers with the ability to choose quantities of each item.

- Bundle

The third composite product type , a bundle relates simple products together to purchase as a single item.

- Virtual

No physical item required for delivery, e.g. services

- Downloadable

A digital rather than physical product. Most product types are implemented as part of the Mage_Catalog module, apart from Mage_Bundle and Mage_Downloadable.

Grouped, Bundle and Configurable products implement a parent-child relationship where a number of other (by default, simple, virtual or downloadable) products get assigned to a main product. This then handles the product data for the whole collection (e.g. group, bundle, or configurable product name, price and status).

Downloadable and Bundle products have extra tables in the database, meanwhile the rest are shared amongst all other product types. Configurable products have an extra table to link to child products, catalog_product_super_link.

Custom Product Type

To create a product type that extends one of the built-in product types, the corresponding product type model should be extended. Otherwise the new product type should extend the Mage_Catalog_Model_Product_Type_Abstract class.

An entry in the module's config.xml is also required:

More complicated products may require other customised areas such as price model and index data retriever.

Price Calculation

When dealing with a single product, the price is always calculated on the fly. The price EAV attribute is loaded with the product and final price is calculated by the price model, Mage_Catalog_Model_Product_Type_Price.

Some product types deal with it differently. In which case they extend this class and implement their own logic. For example, the configurable product overwrites getFinalPrice() and adds additional logic. This custom model can then be specified in config.xml with a <price_model> tag.

Product collections, however, use the price index to retrieve pre-calculated prices, eliminating the need to calculate it for each product.

Final price can be adjusted by the observers of the catalog_product_get_final_price event. By default, only the Mage_CatalogRule module observes this event.

Another method to override produce price is to simply set it on the product. If the price is set, the product will not recalculate it.

Product tier price is separate from normal price (although taken into account when calculating price). It's implemented as a list with a customer group and minimum quantity qualifiers for each tier. Tier prices are displayed in a table, using the catalog/product/view/tierprices.phtml template.

Custom product options get processed when calculating final price. Each option has its own price defined, which gets added to the final price.

Group, tier and special prices all get considered at the same time `($priceModel->getBasePrice())` and the smallest one of the three (or four if you include the regular price) is chosen as the base product price.

Tax

The Mage_Tax module uses a product's tax class and whether or not the product price is inclusive or exclusive of tax in order to identify the correct rate to apply.

The following factors are used to calculate tax on products:

- Product tax class
- The amount of tax already included
- Billing and Shipping addresses
- Customer tax class
- Store settings

Layered Navigation

The classes responsible for rendering the layered navigation are:

- Mage_Catalog_Block_Layer_View

  -Handles the filters and options

- Mage_Catalog_Block_Layer_State

-Controls what is currently being filtered by

To implement layered navigation on attributes with custom source models the Mage_Catalog_Model_Layer_Filter_Abstract::apply() method would need to be overwritten to dictate how the product collection should be filtered.

Layered navigation is rendered by the Mage_Catalog_Block_Layer_View and Mage_Catalog_Block_Layer_State blocks, which use filter blocks for individual filters.

Layered Navigation uses index table for most filters, e.g. price, product attribute index, decimal product index.

Categories

Categories in the Database

Category hierarchy is managed by storing a category's parent id. The full hierarchy is shown in the path column (slash separated IDs). There is a special category with parent_id of 0. This is the true root category and each of the other root categories as defined in Magento use this as a shared parent.

To read and manage a category tree from the database two different classes are used depending if flat catalog is enabled, Mage_Catalog_Model_Resource_Category_Tree and Mage_Catalog_Model_Resource_Category_Flat.

The advantage of flat categories is that it is quicker to query. However, it needs to be rebuilt from the EAV tables each time there is a change.

```
getChildren()
```

returns a comma separated string of immediate children IDs

```
getAllChildren()
```

returns a string or array of all children IDs

```
getChildrenCategories()
```

returns a collection of immediate children categories N.B. If flat catalog is enabled, the only child categories returned will be ones with include_in_menu = 1. In both cases, only active categories are returned.

Catalog Price Rules

Catalog price rules apply discounts to products based on the date, product, website and customer group.

When `getFinalPrice()` is called on a product, the event catalog_product_get_final_price is fired. This is observed by Mage_CatalogRule_Model_Observer which will then look for any catalog price rule that applies to the product. If applicable, it then looks at the database price table and writes the price back to the product model as a Varien data field final_price.

Within the database, the catalogrule table describes rules, their conditions and their actions. catalogrule_product contains the matched products and some rule information. Meanwhile catalogrule_product_price contains the price after the rule has been applied.

Indexing and Flat Tables

Flat catalog tables are managed by catalog indexers. If automatic rebuilding of the indexes is enabled, the catalog indexers get rebuilt every time a product, category or any related entities are updated. The _afterSave() method calls the indexer process. Otherwise they have to be manually re-indexed through admin.

Product type affects price index and stock index where products can define their own custom indexers (in config.xml) to handle their data for these indexes.

The Mage_Index module provides the framework with which custom indexes can be created to help optimise the performance of the site. The Mage_Index_Model_Indexer_Abstract class should be extended to create a new index, implementing the _registerEvent() and _processEvent() methods. Not forgetting to register it in config.xml:

```
<global>
    <index>
        <indexer>
            <{name}>{model}</{name}>
        </indexer>
    </index>
</global>
```

Examples

Mage_Catalog_Model_Product_Type

```
/**
 * Magento
 *
 * NOTICE OF LICENSE
 *
 * This source file is subject to the Open Software License (OSL 3.0)
 * that is bundled with this package in the file LICENSE.txt.
 * It is also available through the world-wide-web at this URL:
 * http://opensource.org/licenses/osl-3.0.php
 * If you did not receive a copy of the license and are unable to
 * obtain it through the world-wide-web, please send an email
 * to license@magentocommerce.com so we can send you a copy immediately.
 *
 * DISCLAIMER
 *
 * Do not edit or add to this file if you wish to upgrade Magento to newer
 * versions in the future. If you wish to customize Magento for your
 * needs please refer to http://www.magentocommerce.com for more information.
 *
 * @category    Mage
 * @package     Mage_Catalog
 * @copyright   Copyright (c) 2012 Magento Inc. (http://www.magentocommerce.com)
 * @license     http://opensource.org/licenses/osl-3.0.php  Open Software License (OSL 3.0)
 */

/**
 * Product type model
 *
 * @category    Mage
 * @package     Mage_Catalog
 * @author      Magento Core Team
 */
class Mage_Catalog_Model_Product_Type
{
    /**
     * Available product types
     */
    const TYPE_SIMPLE       = 'simple';
    const TYPE_BUNDLE       = 'bundle';
    const TYPE_CONFIGURABLE = 'configurable';
    const TYPE_GROUPED      = 'grouped';
    const TYPE_VIRTUAL      = 'virtual';

    const DEFAULT_TYPE      = 'simple';
    const DEFAULT_TYPE_MODEL    = 'catalog/product_type_simple';
    const DEFAULT_PRICE_MODEL   = 'catalog/product_type_price';

    static protected $_types;
    static protected $_compositeTypes;
    static protected $_priceModels;
    static protected $_typesPriority;

    /**
     * Product type instance factory
     *
     * @param   Mage_Catalog_Model_Product $product
     * @param   bool $singleton
     * @return  Mage_Catalog_Model_Product_Type_Abstract
     */
    public static function factory($product, $singleton = false)
    {
        $types = self::getTypes();
        $typeId = $product->getTypeId();

        if (!empty($types[$typeId]['model'])) {
            $typeModelName = $types[$typeId]['model'];
        } else {
            $typeModelName = self::DEFAULT_TYPE_MODEL;
            $typeId = self::DEFAULT_TYPE;
        }
        if ($singleton === true) {
            $typeModel = Mage::getSingleton($typeModelName);
        }
        else {
            $typeModel = Mage::getModel($typeModelName);
            $typeModel->setProduct($product);
        }
        $typeModel->setConfig($types[$typeId]);
        return $typeModel;
    }

    /**
     * Product type price model factory
     *
     * @param   string $productType
     * @return  Mage_Catalog_Model_Product_Type_Price
     */
    public static function priceFactory($productType)
    {
        if (isset(self::$_priceModels[$productType])) {
            return self::$_priceModels[$productType];
        }

        $types = self::getTypes();

        if (!empty($types[$productType]['price_model'])) {
            $priceModelName = $types[$productType]['price_model'];
        } else {
            $priceModelName = self::DEFAULT_PRICE_MODEL;
        }

        self::$_priceModels[$productType] = Mage::getModel($priceModelName);
        return self::$_priceModels[$productType];
    }

    static public function getOptionArray()
    {
        $options = array();
        foreach(self::getTypes() as $typeId=>$type) {
            $options[$typeId] = Mage::helper('catalog')->__($type['label']);
        }

        return $options;
    }

    static public function getAllOption()
    {
        $options = self::getOptionArray();
        array_unshift($options, array('value'=>'', 'label'=>''));
        return $options;
    }

    static public function getAllOptions()
    {
        $res = array();
        $res[] = array('value'=>'', 'label'=>'');
        foreach (self::getOptionArray() as $index => $value) {
            $res[] = array(
               'value' => $index,
               'label' => $value
            );
        }
        return $res;
    }

    static public function getOptions()
    {
        $res = array();
        foreach (self::getOptionArray() as $index => $value) {
            $res[] = array(
               'value' => $index,
               'label' => $value
            );
        }
        return $res;
    }

    static public function getOptionText($optionId)
    {
        $options = self::getOptionArray();
        return isset($options[$optionId]) ? $options[$optionId] : null;
    }

    static public function getTypes()
    {
        if (is_null(self::$_types)) {
            $productTypes = Mage::getConfig()->getNode('global/catalog/product/type')-
>asArray();
            foreach ($productTypes as $productKey => $productConfig) {
                $moduleName = 'catalog';
                if (isset($productConfig['@']['module'])) {
                    $moduleName = $productConfig['@']['module'];
                }
                $translatedLabel = Mage::helper($moduleName)->__($productConfig['label']);
                $productTypes[$productKey]['label'] = $translatedLabel;
            }
            self::$_types = $productTypes;
        }

        return self::$_types;
    }

    /**
     * Return composite product type Ids
     *
     * @return array
     */
    static public function getCompositeTypes()
    {
        if (is_null(self::$_compositeTypes)) {
            self::$_compositeTypes = array();
            $types = self::getTypes();
            foreach ($types as $typeId=>$typeInfo) {
                if (array_key_exists('composite', $typeInfo) && $typeInfo['composite']) {
                    self::$_compositeTypes[] = $typeId;
                }
            }
        }
        return self::$_compositeTypes;
    }

    /**
```

```
     * Return product types by type indexing priority
     *
     * @return array
     */
    public static function getTypesByPriority()
    {
        if (is_null(self::$_typesPriority)) {
            self::$_typesPriority = array();
            $a = array();
            $b = array();

            $types = self::getTypes();
            foreach ($types as $typeId => $typeInfo) {
                $priority = isset($typeInfo['index_priority']) ?
abs(intval($typeInfo['index_priority'])) : 0;
                if (!empty($typeInfo['composite'])) {
                    $b[$typeId] = $priority;
                } else {
                    $a[$typeId] = $priority;
                }
            }

            asort($a, SORT_NUMERIC);
            asort($b, SORT_NUMERIC);

            foreach (array_keys($a) as $typeId) {
                self::$_typesPriority[$typeId] = $types[$typeId];
            }
            foreach (array_keys($b) as $typeId) {
                self::$_typesPriority[$typeId] = $types[$typeId];
            }
        }
        return self::$_typesPriority;
    }
}
```

Describe standard product types (simple, configurable, bundled)

Simple

The Simple Products type should be used for that generally have a single configuration (one-size-fits-all). This might include items such as:

- A Box of Crayons, Small (24 Colors)
- A Box of Crayons, Large (64 Colors)
- SuperHighTech 26” HD Computer Monitor
- Barrack Obama Action Figure (6”)

Grouped

Grouped products allow you to create a new product using one or more existing products in your store. For instance, let’s assume you have a “Barrack Obama Action Figure” and a “George W Bush Action Figure” already in your store and you wanted to sell them as a bundle. You would simply create a new Grouped Product (let’s call it “Obama + Bush (Get Both and Spend Twice as Much!)”, then add both action figures to the group via the “Associated Products” tab.

Note: Unfortunately, you are not able to set a special “group” price directly from the product page. To offer a discount for buying items together, you will need to create a new Shopping Cart Price Rule.

Configurable

Configurable Product : This product enables your customers to select the variant that they want by choosing options. For example, you can sell T-shirts in two colors and three sizes. You would create six simple products as individual products (each with its own SKUs) and then add these six to a configurable product where customers can choose the size and color, and then add it to their cart. Very similar functionality is possible by using Custom Options for Simple products. The difference between a configurable product and a product including custom options is that inventory is not checked or updated for individual options during the purchase of the custom options.

Virtual

Virtual Products are those that do not have a physical or digital counterpart. They do not ship, nor do they have a download link. This product type might be used for services like:

- House Cleaning
- 1-Year Newsletter Subscription

Note: If using Virtual Products for “subscriptions”, it is important to note that there is no built-in way to manage auto-renewing subscriptions. All purchases made in Magento, regardless of Product Type, are one-time purchases.

Bundle

This product type is also known as a “kit” in other eCommerce software. This product type is ideal for circumstances where the user have to select a number of configurable options, but at least one option. This might includes products like:

- Customizable Computer Systems
- Customizable Tuxedos/Suits
- Click here for a video tutorial on using bundles

Downloadable

Downloadable products are similar to virtual products, except that they include the ability to add one or more digital files for download. Files can either be uploaded via the Admin interface, or by uploading directly to the server via FTP and then added by URL. When a customer buys a Downloadable product, Magento will generate a secure, encrypted link (so that the customers can’t see the file’s real location) for that customer to download their file.

This category might include products such as:

- Music/MP3s
- Computer Software

Note: If you have SSL enabled for your site, downloads may fail under all versions of IE as IE contains a bug that prevents downloading over secure connections if the no-cache header is set. This can be easily fixed in an htaccess file by removing the no-cache and no-store headers, or by forcing download links to to be non-secure.

Read Understanding product types online:
https://riptutorial.com/magento/topic/7142/understanding-product-types

Credits

S. No	Chapters	Contributors
1	Getting started with magento	7ochem, Chris Rogers, Community, Gabriel Somoza, goutam, Henry's Cat, Luke Rodgers, Marek Skiba, Pawel Dubiel, RamenChef, Robbie Averill, Stephen Leppik
2	Add different price for multiple store using Magento SOAP API	Harsha Sampath
3	Collections	gebrial, gulshan maurya
4	Complete Product Collection	Twinkal
5	Create Enterprise Gift Cards Programmatically	JPMC, RamenChef
6	Current url	Nolwennig, Twinkal
7	Custom Attributes	Twinkal
8	EAV (Entity Attribute Value)	lalit mohan
9	Get category name from product page	user2925795
10	Get current User	bpoiss, Rickert
11	Get Products from Database	Akif, Alex, baoutch, bpoiss, ctrimm, gulshan maurya, jignesh prajapati, mnoronha, Olavi Sau, pce, SH-
12	Get store name and other details from system configuration	Henry's Cat, Robbie Averill
13	Getting Magento URLs	Nolwennig, Robbie Averill, Steven Church
14	Helpers	Robbie Averill
15	How to Filter	wesleywmd

	Collections	
16	Logging to file	Akif, gulshan maurya, versedi
17	Magento Caching	RamenChef, Robbie Averill, Yogendra - eCommerce Developer
18	Magento error handling, messages and reports	7ochem, Charles, Pankaj Pareek, RamenChef
19	Model	Rickert
20	Module structure	Chris Rogers, RamenChef
21	MVC Structure	lalit mohan
22	Optimizing Magento For Speed	Lemon Kazi
23	Orders	bpoiss, Hemant Sankhla, Luke Rodgers
24	Product Image Urls	Twinkal
25	Quick Task Cheat Sheet	Chris Richardson
26	Rendering	lalit mohan
27	Shell, CLI	djdy, versedi
28	Specific urls	Mayank Pandeyz, Nolwennig, Qaisar Satti, Yogendra - eCommerce Developer
29	Sql script to delete test data	Twinkal
30	Store and Website Data	Twinkal
31	Understanding product types	lalit mohan