Magento Development

Magento Controllers

Code Pools

Magento code pools are stored in app/code/ directory, it consists of:

- core: All the core Magento modules, DO NOT edit core code pools directly as it may break Magento installation due to incompatibilities etc.
- community: Modules by third-party codes, e.g. extensions
- local: Custom made modules, copy core modules here (preserving directory structure) if to modify the core modules instead of modifying core modules directly

Code pool execution priority (lowest to highest, find in next if module not found):

1. local
2. community
3. core
4. `/app/lib`

Namespaces (A.K.A Packages)

Fresh Magento core modules are stored app/code/core directory, "Mage" and "Zend" are 2 namespaces created by Magento. Creating namespaces for your custom modules are just to create a folder in app/code/local directory, it can be any name, e.g. Practice

Naming Conventions

- DO NOT include any "_" (underscore) for folder and file names, as it will be replaced by directory separator (DS) in Magento's autoloader
- Initial caps and camelcase for naming folders and classes
- Use "_" (underscore) in class names to specify path to class files e.g:
  - Magento expect "class Mage_Catalog_Block_Product_Widget_New" in class declaration to find the class file in Mage/Catalog/Block/Product/Widget/New.php

Magento's Autoloaders Class Initialization Steps:

1. "_" (underscores) are replaced by spaces
2. Convert all words to initial caps
3. Spaces are replaced by DS (directory separator)
4. Append .php

e.g. $instance = new Practive_ControllerTest_Model_MyClass() will be converted to Practice/ControllerTest/Model/MyClass.php where the class file is expected

Module Folder Structure

Directory structure for module "ControllerTest" (app/code/local/Practice/ControllerTest/) should contain the following directories:

- Block/
- controllers/
  - Optional as not all modules contain controllers
- etc/
  - Store module configuration and system files (expects .xml extension)
- Helper/
- Model/
- sql/

Configuration Files (Module Configuration)

- File can be any names as long as it end with xml
- Recommended naming conventions: Namespace_ModuleName.xml
- System wide configuration stored in app/etc/modules/ directory

Configuration Steps:

1. Create and edit app/etc/modules/Practice_ControllerTest.xml where Magento expects to find the module (ControllerTest) main config file in `app/local/Practice/ControllerTest/etc/config.xml`:

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

```
    <modules>
        <Practice_ControllerTest>
           <active>true</active>
           <codepool>local</codepool>
        </Practice_ControllerTest>
    </modules>
</config>
```

2. Create and edit app/code/local/Practice/ControllerTest/etc/config.xml:

```
<?xml version="1.0"?>
<config>
    <modules>
            <Practice_ControllerTest>
                 <version>0.0.1</version>
        </Practice_ControllerTest>
    </modules>
</config>
```

3. To define a controller, add the following code between "" tags after "" tags in `app/code/local/Practice/ControllerTest/etc/config.xml`:

```
<frontend>
    <routers>
        <test_controller>
            <module>Practice_ControllerTest</module>
            <frontName>requestflowtest</frontName>
        </test_controller>
    </routers>
</frontend>
```

- frontend => Specify an "area", possible values: frontend, backend or global
- routers => A role
- test_controller => Unique controller config

Creating a Controller

Create and edit `app/code/local/Practice/ControllerTest/controllers/IndexController.php`

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

Testing

Navigate to "//requestflowtest"

Routing

- To specify as admin router, use "admin" between the "" tags in config.xml, causing it to route for admin interface only
- To specify as frontend router, use "frontend" between "" tags in config.xml, causing it to route for frontend interface only

Order:

1. Admin
2. Standard
3. Cms
4. Default

Sample URL: http:////catalog/product/view/id/1

- catalog: frontName tag in config.xml
- product: ProductController
- catalog/product results: Mage/Catalog/controllers/ProductController.php
- view
  - viewAction() method in the controller
  - If viewAction() method not found, assume indexAction() in IndexController

---

Magento View

Layouts

Magento uses xml files to specify the layout, available handles for

.xml includes: 1. label *Specify the title or label to be displayed to user 2. reference* Add an existing block to the page *Attribute "name" specifies the existing block name* action *block 3. remove* Remove an existing block from the layout *Attribute "name" specifies the block to remove*

e.g. 4. update *Load specified layout handle into the current layout handle, provides a kind of inheritance to the current layout handles* Can be use to paste an entire layout handle into the current ones *Attribute "handle" must be included which specifies the layout handle* Syntax: 5. block *type/class* Mandatory, sets an identifier of a PHP class name *name* Specifies the name of the block, can be reused for the reference directives *template* Specifies which .phtml file to render the page *as* Used to assign an alias for the block *If specified, this block can be used in the $this->getChildHtml('alias') method* output *Specifies the method name the system will use to gather an output instead of the default toHtml() method* Only required for the root block hierarchy because child blocks can be obtained through the getChildHtml() method *parent* Override the default parent block *before* Used to specify that the current block is before the specified block *after Used to specify that the current block is after the specified block 6. action* method *Specifies the method name that should be called* block *ifconfig* Allows checking of the configuration parameter before the action processor *json* Specifies that the data has to be decoded before passing into the methods *translate* Specifies which identifier in the child node must be passed for processing for the translation procedure *module* Specifies action module *helper* Specifies the helper functions, allow processing of data
Magento stores all .xml files in app/design/[areaname]/[packagename]/[themename]/layout folder
### Block Blocks are just a set of PHP classes used in rendering. Block types: *Mage_Core_Block_Template (core/template)* Render template block defined by its template attribute *Mage_Core_Block_Text_List (core/text_list)* All child blocks are rendered automatically without the need to call GetChildHtml() method *Mage_Core_Block_Messages (core/messages)* Renders success or notice messages *Mage_Core_Block_Template_Links (page/template_links)* Used to create a list of links *Mage_Core_Block_Switch (page/switch)* Render a store switcher ---
## Retrieve Categories from Other Stores `**Using App Emulation**`

```
    $appEmulation = Mage::getSingleton('core/app_emulation');
    $store_id = 1; // The ID if your store
    $initialEnvironmentInfo = $appEmulation->startEnvironmentEmulation($store_id);

    $children = Mage::getModel('catalog/category')->load(306)->getChildrenCategories();
    foreach($children as $child){
       echo $child->getName() . " " . $child->getUrl() . "<br/>";
    }

    $appEmulation->stopEnvironmentEmulation($initialEnvironmentInfo);
```

--- ## Magento New Product Widget by Store 1. Create directory structure: `mkdir -p app/code/local/Mage/Catalog/Block/Product/Widget` 2. Copy `app/code/core/Mage/Catalog/Block/Product/Widget/New.php` to `app/code/local/Mage/Catalog/Block/Product/Widget`:

```
cp
app/code/core/Mage/Catalog/Block/Product/Widget/New.php to
    app/code/local/Mage/Catalog/Block/Product/Widget
```

3. Modify `New.php` `_getRecentlyAddedProductsCollection()` function and add the following lines:

```
$_rootcatID =
Mage::app()->getStore()->getRootCategoryId();

    $collection = $this->_addProductAttributesAndPrices($collection)
       ->joinField('category_id', 'catalog/category_product', 'category_id',
'product_id=entity_id', null, 'left')
       ->addAttributeToFilter('category_id', array('in'  => $_rootcatID))
       ->addAttributeToSelect('*')
```

--- ## Add new field in magento(1.9) customer registration You need to create a new extension to make it clean.

Let's call the extension `StackExchange_Customer`.
You will need the following files: `app/etc/modules/StackExchange_Customer.xml` - the declaration file <?xml version="1.0"?> truelocal`app/code/local/StackExchange/Customer/etc/config.xml` - the configuration file: <?xml version="1.0"?> 1.0.0StackExchange_Customer_Helper StackExchange_CustomerMage_Customer_Model_Resource_Setupstackexchange_customer.xml StackExchange_Customer.csv `app/code/local/StackExchange/Customer/sql/stackexchange_customer_setup/install-1.0.0.php` - the install file. Will add the new attribute. <?php $this->addAttribute('customer', 'license_number', array('type' => 'varchar', 'label' => 'License Number', 'input' => 'text', 'position' => 120, 'required' => false,//or true 'is_system' => 0,)); $attribute = Mage::getSingleton('eav/config')->getAttribute('customer', 'license_number'); $attribute->setData('used_in_forms', array('adminhtml_customer', 'checkout_register', 'customer_account_create', 'customer_account_edit',)); $attribute->setData('is_user_defined', 0); $attribute->save(); `app/code/local/StackExchange/Customer/Helper/Data.php` - the module main helper <?php class StackExchange_Customer_Helper_Data extends Mage_Core_Helper_Abstract { } This will add your attribute for the customer.
It should work nicely on the backend.
Unfortunately you have to edit the frontend templates manually now because Magento does not have any event or empty block where you can put your fields.
For this you need the following. `app/design/frontend/base/default/layout/stackexchange_customer.xml` <?xml version="1.0"?> stackexchange_customer/form/edit.phtml stackexchange_customer/register.phtml

And now the templates.

`app/design/frontend/base/default/template/stackexchange_customer/register.phtml` - the registration template.
For this one make a clone of the `/app/design/frontend/{package}/{theme}/template/persistent/customer/form/register.phtml` and just insert this somewhere inside the form. I don't need to post the full file here. Arrange it as you please

```
<li>
    <label for="license_number"><?php echo $this->__('License Number') ?></label>
    <div class="input-box">
        <input type="text" name="license_number" id="license_number" value="<?php echo $this->escapeHtml($this->getFormData()->getLicenseNumber()) ?>" title="<?php echo $this->__('License Number') ?>" class="input-text" />
    </div>
</li>
```

`/app/design/frontend/base/default/template/stackexchange_customer/form/edit.phtml` For this one clone `/app/design/frontend/{package}/{theme}/template/customer/form/edit.phtml` and insert somewhere inside the form this:

```
<li>
    <label for="license_number"><?php echo $this->__('License Number') ?></label>
    <div class="input-box">
        <input type="text" name="license_number" id="license_number" value="<?php echo $this->htmlEscape($this->getCustomer()->getLicenseNumber()) ?>" title="<?php echo $this->__('License Number') ?>" class="input-text" />
    </div>
</li>
```

You can also create the translation file. Is not mandatory but it's nice to have `app/locale/en_US/StackExchange_Customer.csv` `"License Number","License Number"`

Clear the cache and you should be set.

How to delete an eav product attribute programmatically

It is my view that this sort of thing should be done via set-up script or if that is not possible then via the admin section. I cannot think of a situation where you need to write a script for this, but maybe there is one.

For a set-up script it is incredibly simple:

```
$installer = $this;
$installer->startSetup();
// Remove Product Attribute
$installer->removeAttribute('catalog_product', 'product_attribute_code');
// Remove Customer Attribute
$installer->removeAttribute('customer', 'customer_attribute_code');
$installer->endSetup();
```

If you need something more complex like removing groups or attribute sets that can also be done via a script too.

Fix Magento Error "dbModel read resource does not..."

We came across the following magento error this morning whilst conducting a database rollback for a client of ours. "dbModel read resource does not implement Zend_Db_Adapter_Abstract"

The solution to solving this is actually quite simple, clear the folders/files within the magento_root /var/cache/ folder.

Error 503 Admin and Frontend

Login to webroot and remove the file maintenance.flag

Display Stock Level on Product Details Page

Edit `app/design/frontend/<theme>/<template>/template/catalog/product/view/type/availability/default.phtml`:

```
<span class="value">
<?php
$quantity = Mage::getModel('cataloginventory/stock_item')->loadByProduct($_pro
duct)->getQty();
if ($quantity > 0)
     echo intVal($quantity);
else
     echo $this->helper('catalog')->__('In stock');
?>
</span>
```

##Adding Module Dependencies

In `app/etc/modules/Practice_OneStepCheckout.xml` add a depends node under `Practice_OneStepCheckout` tags:

```
<Practice_OneStepCheckout>
    <depends>
        <Mage_Sales />
        <Mage_CatalogInventory />
        <Mage_Checkout />
    </depends>
</Practice_OneStepCheckout>
```

Adding Custom Scripts to Magento

You can set different scripts per website or even store views:

Use the field Miscellaneous Scripts from System->Configuration->Design->Head and put your scripts in there. They will be added before the tag and you can set different scripts per website or even store views.

Adding Static Block to .phtml

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

Where `static_block_identifier` is the identifier for the CMS static block "identifier"

Revision #1
Created 29 April 2017 09:05:05 by Tingwai
Updated 29 April 2017 09:05:30 by Tingwai