

## Exercise Solutions

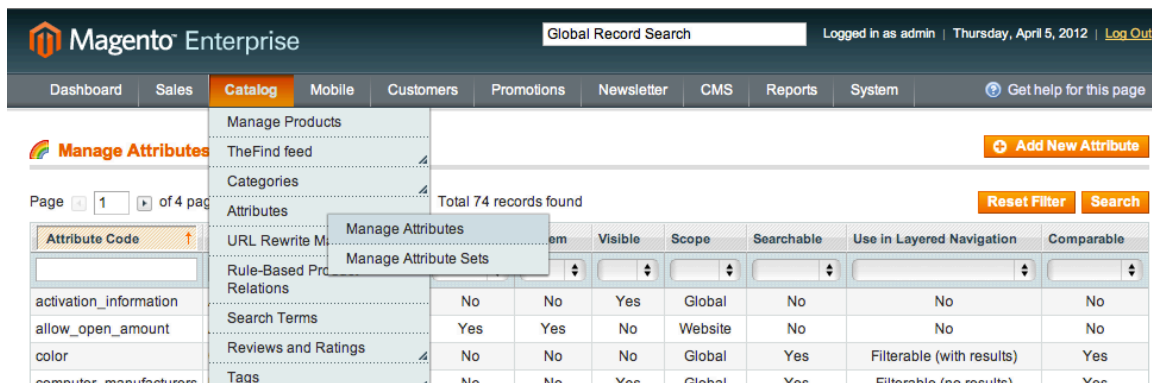
### Section 6 Lesson 3. EAV Attribute Management

#### Exercise 1: Create a Text Input Attribute from the Admin Interface

Use your browser to do this exercise.

##### Step 1. Add a text input attribute to the *Default* attribute set

- Open Catalog > Attributes > Manage Attributes.
- Choose "New Attribute" from the upper right corner.



The screenshot shows the Magento Enterprise Admin Interface. The top navigation bar includes 'Dashboard', 'Sales', 'Catalog', 'Mobile', 'Customers', 'Promotions', 'Newsletter', 'CMS', 'Reports', and 'System'. The 'Catalog' menu is expanded, showing 'Manage Products', 'TheFind feed', 'Categories', 'Attributes', 'Manage Attributes', and 'Manage Attribute Sets'. The 'Attributes' menu item is selected, leading to the 'Manage Attributes' page. The page displays a table of attributes with columns: Attribute Code, Name, Visible, Scope, Searchable, Use in Layered Navigation, and Comparable. The table shows several attributes, including 'activation\_information', 'allow\_open\_amount', 'color', and 'computer\_manufacturers'. The 'Add New Attribute' button is located in the top right corner of the page.

- Configure the attribute as follows:

**Attribute Code:** training\_attribute

**Scope:** Global

**Catalog Input Type for Store Owner:** Text Field

**Label:** Training Attribute

**Visible on Product View Page on Front-end:** Yes

- Open Catalog > Attributes > Manage Attribute Sets. Add the attribute to the *Default* set by dragging it from the *Unassigned Attributes* area to the *General* group on the left. Note that you may need to scroll the div containing the unassigned attributes to see the attribute you created.

### Step 2. Check that the attribute appears on the product edit page

- Edit a product associated with the *Default* attribute set. (You can use the filter in the product grid.) For example, one product associated with the *Default* attribute set from the sample data is the *Universal Camera Charger*.

Notice how the attribute's input is displayed in the order and group in which it was placed.

### Step 3. Make the attribute visible on the frontend product view page

- a) To display the attribute value on the product view page in the frontend, be sure the value of the attribute property "Visible on Product View Page on Front-end" is set to Yes.
- b) View the frontend page for the product which was edited in step 2; the value should display in the "Additional Information" tab.

## Exercise 2: Create a Text Input Attribute from a Setup Script

Use your IDE to do this exercise. Do this exercise and the following ones by expanding the module `Training_Practice`.

### Step 1. Follow the steps from Exercise 1 and create a text input attribute with a setup script

- a) Add a setup resource and a module version number to the `Training_Practice` module's `etc/config.xml` file. Refer to the exercise solutions from Section 5, Lesson 2 for instructions how to do so.

Use the setup resource name `training_practice_setup` and the version `0.0.1`.

- b) Create an `install-0.0.1.php` file in the `sql/training_practice_setup` folder.
- c) Because we are adding an attribute to the `catalog_product` entity it is important to use the catalog module's setup class.

```
<?php

/* @var $installer Mage_Catalog_Model_Resource_Setup */
$installer = Mage::getResourceModel('catalog/setup', 'default_setup');

$installer->startSetup();

$installer->endSetup();
```

Using the appropriate module setup file ensures that the proper additional attribute table is used in addition to `eav_attribute`, in this case in the table `catalog_eav_attribute`. Note that setup classes require a connection resource name (`default_setup`) to be passed to their constructor.

- d) Next, as a matter of convenience, assign various configuration parameters to variables.

```
<?php

/* @var $installer Mage_Catalog_Model_Resource_Setup */
$installer = Mage::getResourceModel('catalog/setup', 'default_setup');

$installer->startSetup();

$attrCode = 'script_attribute'; //unique for the given entity
$entityCode = Mage_Catalog_Model_Product::ENTITY; //catalog_product

$config = array(
    'type' => 'text',
    'label' => 'Script Attribute',
    'visible_on_front' => true,
    'user_defined' => true,
);

//...

$installer->endSetup();
```

- e) Then use the EAV setup method `addAttribute()` to create the attribute, and the method `addAttributeToGroup()` to assign it to the *General* group in the *Default* attribute set.

```
<?php

/* @var $installer Mage_Catalog_Model_Resource_Setup */
$installer = Mage::getResourceModel('catalog/setup', 'default_setup');

$installer->startSetup();

$attrCode = 'script_attribute'; //unique for the given entity
$entityCode = Mage_Catalog_Model_Product::ENTITY; //catalog_product

$config = array(
    'type' => 'text',
    'label' => 'Script Attribute',
    'visible_on_front' => true,
    'user_defined' => true,
);

$installer->addAttribute($entityCode, $attrCode, $config);

$installer->addAttributeToGroup(
    $entityCode, 'Default', 'General', $attrCode
);

$installer->endSetup();
```

- f) Clear the configuration cache if necessary, hit any page, and verify that the attribute is set up in the Admin and displaying on the frontend.

### Exercise 3: Create a Multiselect Product Attribute from the Setup Script

Use your IDE to do this exercise.

#### Step 1. Create a multiselect product attribute

- a) Create an upgrade script *file upgrade-0.0.1-0.0.2.php* in *sql/training\_practice\_setup/*.

- b) Create a similar script as before. Note the changes in the attribute configuration array.

```
<?php

/* @var $installer Mage_Catalog_Model_Resource_Setup */
$installer = Mage::getResourceModel('catalog/setup','default_setup');

$installer->startSetup();

$attrCode = 'multi_attribute'; //unique for the given entity
$entityCode = Mage_Catalog_Model_Product::ENTITY; //catalog_product

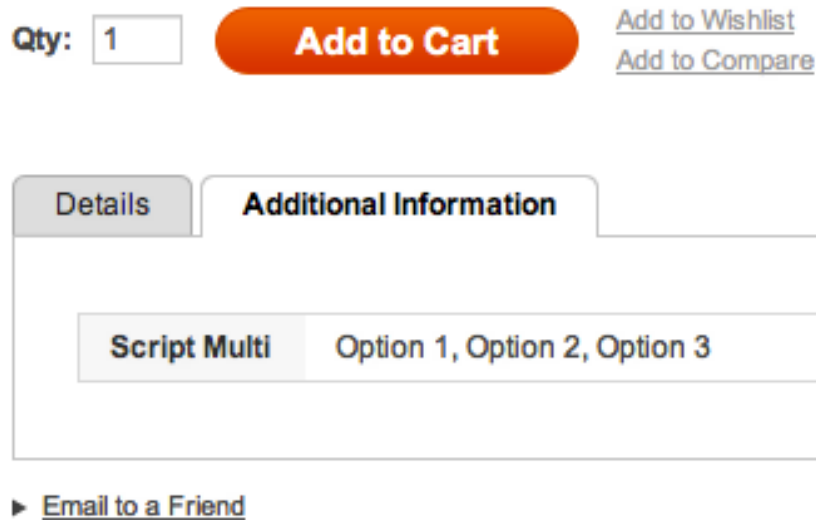
$config = array(
    'type'=> 'text',
    'input' => 'multiselect',
    'label'=> 'Script Multi',
    'backend' =>
    'eav/entity_attribute_backend_array',
    'visible_on_front' => true,
    'required' => false,
    'option' => array(
        'values' => array(
            'Option 1',
            'Option 2',
            'Option 3'
        )
    ),
);

$installer->addAttribute($entityCode, $attrCode, $config);

$installer->endSetup();
```

**Note:** This time the attribute is not `user_defined`, and because of that it is added to the *General* group of every attribute set automatically.

- c) Change the `Training_Practice` module's version number to `0.0.2`.
- d) Clear the configuration cache, reload any page, and verify that the attribute has been created. You can do so by visiting the attribute management page or by editing a product.
- e) Edit a product and select two or more options for the `multi_attribute` attribute. View the attribute display on the product page in the frontend. Note that the values are displayed as comma-separated values in the frontend.



---

**Note:** The rendering of these values is occurring through the default frontend model's `getValue()` method from its superclass, `Mage_Eav_Model_Entity_Attribute_Frontend_Abstract`.

We'll change this behavior in the next lesson by changing the model, which will be handling the rendering of our custom multiselect attribute values.

---

## Exercise 4: Customize the Frontend Rendering of the Attribute Values

Use your IDE to do this exercise.

### Step 1. Customize the rendering of the values of the multiselect product attribute created in the previous exercise

- Create the following in a new upgrade script named `upgrade-0.0.2-0.0.3.php` with the following content.

```

<?php
/* @var $installer Mage_Catalog_Model_Resource_Setup */
$installer = Mage::getResourceModel('catalog/setup','default_setup');

$attrCode = 'multi_attribute'; //unique for the given entity
$entityCode = Mage_Catalog_Model_Product::ENTITY; //catalog_product

$installer->startSetup();

//set the attribute as allowing HTML to render in the frontend
$installer->updateAttribute(
    $entityCode,
    $attrCode,
    'is_html_allowed_on_front', //a `catalog_eav_attribute` table column
    true
);

//set custom model to format the return value from the database
$installer->updateAttribute(
    $entityCode,
    $attrCode,
    'frontend_model', //an `eav_attribute` table` column
    'training_practice/entity_attribute_frontend_html_list'
);

$installer->endSetup();

```

Two items are noteworthy in this upgrade script:

- The first note is regarding the parameter `frontend_model`. The column name passed as the third `updateAttribute()` parameter is *not* mapped through `_prepareValues()`, so it *must* match the column name from the `eav_attribute` table. This is not the case for the column names passed in the `addAttribute()` configuration array, as those *are* mapped through the `_prepareValues()` method.
- The second note is in regards to the use of custom backend, source, and frontend models in attributes. If the class specified in any of these `eav_attribute` fields cannot be loaded (because the module has been deleted, inactivated, etc.), then the system will likely throw a fatal exception when the attribute is loaded (often triggered via a product load). This means that to fully uninstall a module or an attribute that uses custom attribute models, one must delete the attribute (or its source/backend/frontend models) from the `eav_attribute` table.

- b) Based on the model class group `training_practice`, and the `frontend_model` property specified above, the frontend model class needs to be defined in the file *Training/Practice/Model/Entity/Attribute/Frontend/Html/List.php*. Create this file and define the class as follows.

```
class Training_Practice_Model_Entity_Attribute_Frontend_Html_List
    extends Mage_Eav_Model_Entity_Attribute_Frontend_Abstract
{
    public function getValue(Varien_Object $object)
    {
        //formatting code and logic will go here
    }
}
```

The frontend model class must extend from the frontend abstract class and overrides only the `getValue()` method.

- c) Next, add the method definition code shown below.

```
class Training_Practice_Model_Entity_Attribute_Frontend_List_Html
    extends Mage_Eav_Model_Entity_Attribute_Frontend_Abstract
{
    public function getValue(Varien_Object $object)
    {
        // 1) retrieve value
        $value = $object->getData(
            $this->getAttribute()->getAttributeCode()
        );

        // 2) check the attribute input type for suitability
        if ($this->getConfigField('input') == 'multiselect'
            && is_array($this->getOption($value))
        ){
            $output = '<ul><li>';
            $output .= implode('</li><li>', $value);
            $output .= '</li></ul>';
        } else {
            // 3) sanity check in case this model's logic does not
            $output = parent::getValue($object);
        }

        return $output;
    }
}
```

- d) Change the module version number to `0.0.3`, clear the cache if necessary, and refresh the product page. The attribute values should now be displayed in a list.



Qty:  [Add to Cart](#) [Add to Wishlist](#) [Add to Compare](#)

Details

Additional Information

Script Multi

Option 1  
Option 2  
Option 3

► [Email to a Friend](#)

## Exercise 5: Create a Select Attribute with a Predefined List of Options

Use your IDE and browser to do this exercise.

In the Admin, navigate to **Customer > Manage Customers** and choose a customer to edit (or create a customer if there are none). Click on the Account Information tab. This is where the new attribute will be added.

[Dashboard](#)
[Sales](#)
[Catalog](#)
[Mobile](#)
[Customers](#)
[Promotions](#)
[Newsletter](#)

Customer Information

Customer View

Account Information

Addresses

Orders

Billing Agreements

Recurring Profiles (beta)

RMA

Shopping Cart

Wishlist

**Test Member**

[Back](#)
[Reset](#)
[Create Order](#)
[Delete](#)

Account Information

Associate to Website \*

Created From

Customer Group \*

Prefix

First Name \*

Main Website

Admin

Store Special

Test

### Step 1. Create a new customer attribute 'priority' using a setup script

- a) Create a new setup script *upgrade-0.0.3-0.0.4.php*. For this script, use the customer module's setup class.

```
<?php

/* @var $installer Mage_Customer_Model_Resource_Setup */
$installer = Mage::getResourceModel('customer/setup', 'default_setup');

$installer->startSetup();

$installer->endSetup();
```

- b) Next, add the code to install the attribute on the customer entity using `addAttribute()`.

```
<?php

/* @var $installer Mage_Customer_Model_Resource_Setup */
$installer = Mage::getResourceModel('customer/setup', 'default_setup');

$attrCode = 'priority';
$entCode = 'customer';

$installer->startSetup();

$config = array(
    'label'=>'Priority',
    'type'=>'int',
    'input'=>'select',
    'source'=>'training_practice/entity_attribute_source_priority'
);

$installer->addAttribute($entCode, $attrCode, $config);

$installer->endSetup();
```

---

**Note:** If the script were executed at this point, the attribute would be installed but would not be added to the customer form configuration table.

---

- c) In order to add the attribute to the Admin form, retrieve the attribute model for the new `priority` attribute using the `eav/config` singleton. The attribute model instance is used to save the form configuration.

```

<?php
//snip...
$config = array(
    'label'=>'Priority',
    'type'=>'int',
    'input'=>'select',
    'source'=>'training_practice/entity_attribute_source_priority'
);

$installer->addAttribute($entCode, $attrCode, $config);

Mage::getSingleton('eav/config')
    ->getAttribute('customer', 'priority')
    ->setUsedInForms(array('adminhtml_customer'))
    ->save();

$installer->endSetup();

```

- d) Change the module version number to 0.0.4, clear the configuration cache if necessary, and refresh the current view of the customer in the Admin. The Priority field should be displayed at the top.

The screenshot shows the Magento Admin interface. The top navigation bar includes 'Dashboard', 'Sales', 'Catalog', 'Mobile', 'Customers' (highlighted), 'Promotions', 'Newsletter', and 'CMS'. Below this, the 'Customer Information' section is active, showing a sidebar with links: 'Customer View', 'Account Information', 'Addresses', 'Orders', 'Billing Agreements', 'Recurring Profiles (beta)', 'RMA', 'Shopping Cart', and 'Wishlist'. The main content area displays the 'Test Member' profile with buttons for 'Back', 'Reset', 'Create Order', and 'Delete Customer'. The 'Account Information' form includes fields for 'Priority \*' (set to 9), 'Associate to Website \*' (Main Website), 'Created From' (Admin), 'Customer Group \*' (Store Specials Member), and 'Prefix'.

The `used_in_forms` property is evaluated in the `Mage_Customer_Model_Resource_Attribute::_afterSave()` method. The data is saved in the `customer_form_attribute` table.