Magento Extension Developer's Guide

Table of Contents

Table of Figures

Magento Overview

Overview

Magento is a feature-rich eCommerce platform built on open-source technology that provides on-line merchants with unprecedented flexibility and control over the look, content and functionality of their eCommerce store. Magento's intuitive Administration interface features powerful marketing, search engine optimization and catalog-management tools to give merchants the power to create sites that are tailored to their unique business needs.

Magento is an MVC-based application written in PHP which is divided into groups of modules. Each module separates areas of functionality from others and helps to minimize dependencies between itself and other modules. The following is a diagram of the Magento core application structure:

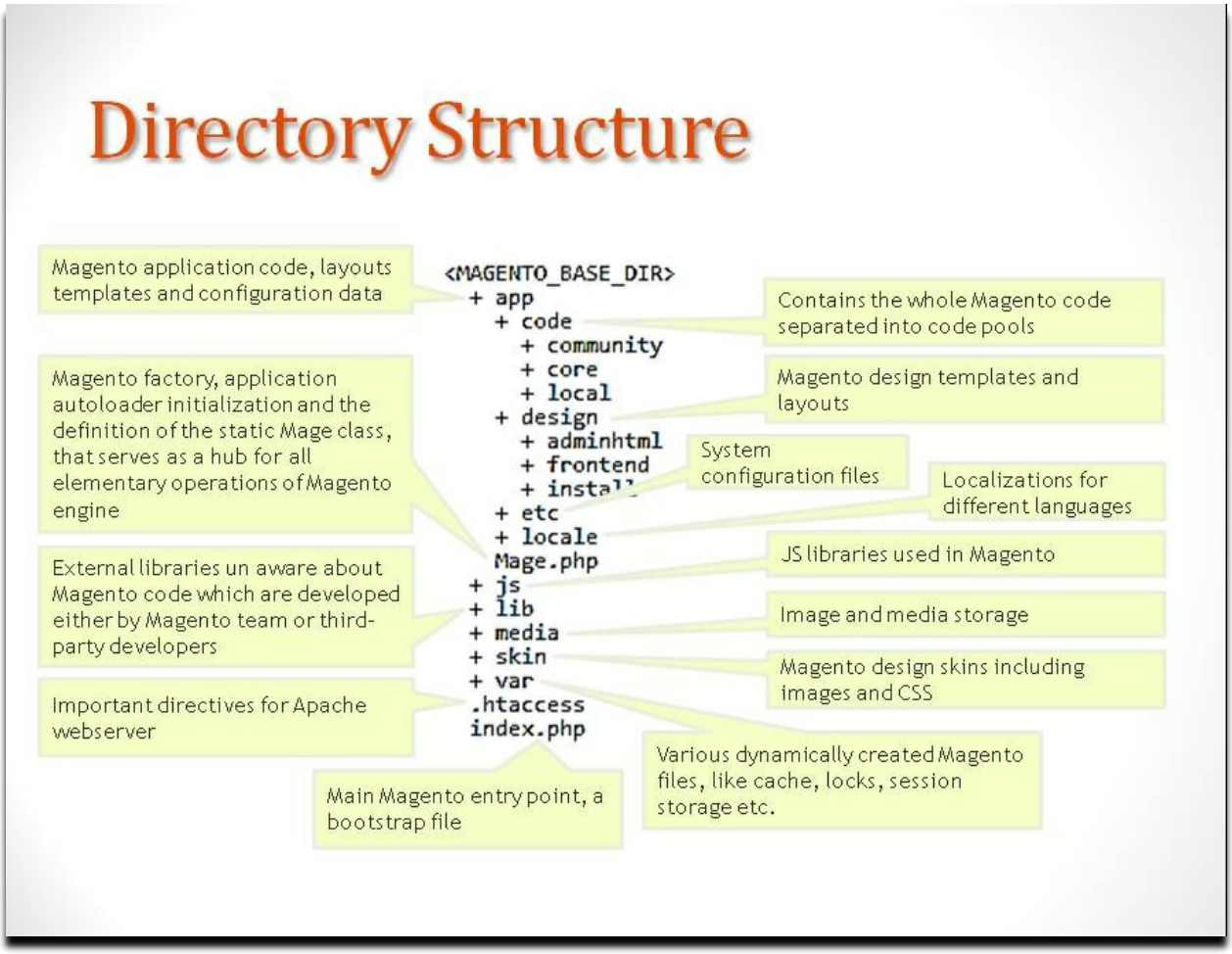

Figure 1: Magento core application structure

Magento is an MVC-based application which means its code and design elements are separated. Also, Magento is **configuration-based**, which means it uses configuration files rather than naming conventions for its settings.

Code

All code files are grouped together based on functionality called **modules,** in Magento.

Modules are the core of Magento. Every action on the site, frontend or backend, goes through a module. Modules act as containers for one or more of the following:

- Settings
- Database schemas
- Database data
- Rendering objects
- Utility helpers
- Data models
- Action controllers.

A module can be made of all seven of these things, or just one.

Modules are further grouped into **code pools**, which allow for safe and easy customization and extending of its core functionality. Using code pools, developers ensure that native functionality can be upgraded with new releases of the core product, and that it is protected from any changes that a merchant wants to make to the platform.

Code Pools

Magento contains three code pools:

- The **`core` code pool** contains the base Magento application. Only Magento core product developers should change files in this code pool.
- The **`community` code pool** contains files that have been extended from the core code pool and that can apply to several different instances of Magento. Most extension developers make their changes and additions in this code pool because their extensions can be used by any merchant who installs their extension.
- The **`local` code pool** contains files that have been extended from the core code pool but that will only be used by a specific instance of Magento. The local code pool is used by merchants who want to customize their platforms, but do not want to make their customizations available to the community at large.

Module code structure

As mentioned above code files are grouped together based on functionality **into modules.** Module can consist of the following components:

- **Blocks**: Responsible for preparing data for display in templates
- **Models**: Implement business logic
- **Resource Models**: Provide an access layer to data for the extension
- **Controllers**: Contain methods called actions which process web server requests that correspond to a specific module.
- **Configuration**: Module-specific XML configuration files that tell the application how the module interacts with it
- **Helpers:** Auxiliary classes that implement commonly used business logic. Examples: forms, validators, formatters
- **Schema SQL Scripts**: SQL scripts that implement schema changes to a specific module
- **Data SQL Scripts**: Scripts that manipulate the data for the SQL schema of a module. Pay strict attention to this as different upgrades were added in Magento Enterprise Edition (EE) 1.11 and Magento Community Edition (CE) 1.6

Magento Coding Standards

We consider Magento's coding standards to be required prerequisite reading for any developers who wish to work with the platform.

Design

To exactly control the look and feel of each store in your Magento installation, Magento allows you to create **themes**. Related themes are grouped together in design packages. Each store in your Magento installation can use its own theme, or they can all share a single theme, or some combination of each.

A design package is a collection of related themes. You can have any number of design packages installed, but there must always be at least one. When installed, Magento has a special package named the “base package.”

Themes inside a design package contain the actual files that determine the visual output and frontend functionality of your store. A theme can belong to only one design package. By convention, each design package must contain a default theme. Additionally, a design package can contain any number of variations on that default theme - called variously non-default themes or theme variants.

Themes

A Magento theme is composed of different file types (layout, template, locale) and/or skin files (CSS, images, theme-specific JavaScript) that create the visual experience of your store. These files reside in two main directories in your Magento file system:

- `app/design` directory – Files that control how the page templates are rendered
- `skin` directory – Files that control the visual aspects of the theme---CSS, images, etc.

Theme files are organized into the following subdirectories:

- **Layout:** Contains the basic XML files that define block structure for different pages as well as control meta information.
- **Template:** Contains the PHTML files that contain xHTML markups and any necessary PHP to create logic for visual presentation. Some templates are page templates and some are block templates.
- **Locale:** Contains simple CSV text documents organized on a per language basis containing translation strings (as name-value pairs) for all text produced by Magento (e.g., for interface elements and messages, not products and categories)

Skin files are organized into the following subdirectories:

- **CSS:** Contains the CSS files used to control visual styling of the website
- **Images:** Contains all images used by the theme
- **JS:** Contains theme-specific JavaScript routines and callable functions.

Theme Fall-back

Magento's theme hierarchy allows developers to localize their themes, changing or adding only those files that are actually necessary for a custom theme. Any files that are not customized in the new theme are pulled from the default theme and base package.

1. If a custom theme is specified, the application looks for the requested file in:
    1. `app/design/frontend/`*`custom_package`*`/`*`custom_theme`*
    2. `skin/frontend/`*`custom_package`*`/`*`custom_theme`*
2. If the file is not found in the custom theme, the application moves up a level and looks for it in:
    1. `app/design/frontend/`*`custom_package`*`/default`
    2. `skin/frontend/`*`custom_package`*`/default`
3. If not found in the default , the application looks for the requested file in:
    1. `app/design/frontend/base/default`
    2. `skin/frontend/base/default`
4. If not found, a rendering error will occur.

Both a design package and theme can be assigned on either the website level and/or store view level through the Magento Admin panel.

Configuration

Modules are defined as being on or off in an XML configuration system located in `app/etc/modules/`. Those configuration files also indicate which code pool the module is placed in, and which modules it depends on.

Each module can specify its own settings in an XML file as well, located under the module's `etc/` directory.

An Example Magento Extension

Extension overview

We will illustrate best practices and restrictions for developing extensions using an example "News Management" extension. This extension will allow a merchant to manage news posts in the Admin panel (News > Manage News) and then view the posts on the frontend using a link in the footer. If you want to skip this example, you can see the recommendations in list format in Appendix A at the end of this document.

Getting Started: Code Setup and Configuration

Selecting our code pool

Before we start writing code, we have to prepare its structure. First we need to decide which code pool our extension will be stored in.

Extension Best Practices

- Code for extensions that are intended for community use should be stored in the `community` code pool.
- In order to improve readability and make the relationship between templates and the module code more clear, templates should be stored in separate folders that are named with the module namespace and name. For example, in our extension we will create the folder structure for frontend templates so that the folder `magentostudy` will include the folder `news` where all our templates will be placed. And we place the newly created folder `magentostudy` into `frontend/base/default/template` according to the restrictions above.
- For similar reasons, layouts should be named with the module namespace and name. For example, in our extension we will name our frontend layout `magentostudy{{_news.xml}}` and place it into `frontend/base/default/layout`.
- Be advised that native layouts don't include a namespace in their names. For example, the layout related to the core module `Mage_Catalog` is named just `catalog.xml`, not `mage_catalog.xml`

Next, we need to decide on a namespace and a name for our extension. We will use a namespace of `Magentostudy` and we will name our extension `News`.

Creating our code structure

Now that we have chosen a code pool, namespace, and extension name let's start creating its basic file/folder structure for our code. Whether or not you include the other folders depends on what your extension does. For example, if your extension doesn't need any controllers, you won't need the `controller` folder.

See the following diagram for our extension folder structure:

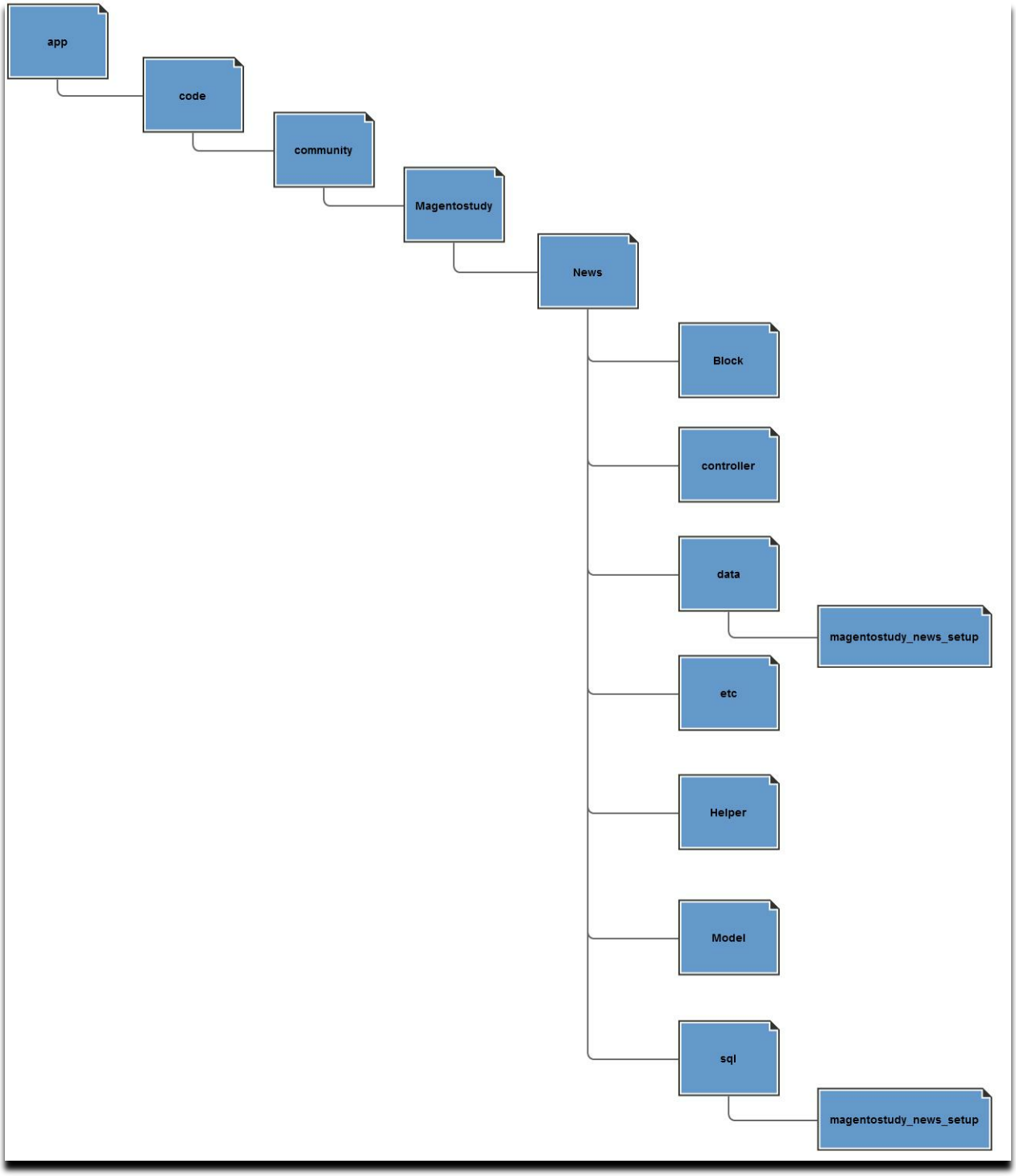

Figure 2: Our extension folder structure

Creating our configuration files

Next, we need to let Magento know that our module exists and is active. We also need to configure the module's content. To do this we need to create two configuration files, an application-specific file and a module-specific file:

Initial configuration file	Module configuration file
• Located in `app/etc/modules` which already exists in native Magento • Mandatory • Is named by module namespace and name (`Magentostudy_News.xml`) • Tells Magento which code pool our extension is stored in and whether it's enabled or not. • Also tells Magento which modules our extension depends on to function properly.	• Located in `app/code/<code pool>/<namespace name>/<module name>/etc/` • Mandatory • Named `config.xml` • Specifies which blocks, models, helpers, routes, etc the extension will contain.

Application configuration XML

Because we know our extension will be stored in the `community` code pool and that it will be active, the only thing that is left to decide is which modules it will depend on. We do not plan on rewriting any existing core modules, but we do want to add our extension to the Admin panel. To allow for this, our module will depend on the core `Mage_adminhtml` module.

Extension Best Practices

- Always be sure to check your module's dependencies because a missing or incorrectly defined dependency can cause your extension to produce fatal errors.
- Only necessary dependencies should be specified because they influence the loading sequence of the application's core modules. Unnecessary dependencies should not be included

Magentostudy_News.xml

```
<?xml version="1.0"?>
<!--
/**
 * Module initial config
 *
 * @author Magento
 */
-->
<config>
    <modules>
```

```
        <Magentostudy_News>
            <active>true</active>
            <codePool>community</codePool>
            <depends>
                <Mage_adminhtml />
            </depends>
        </Magentostudy_News>
    </modules>
</config>
```

Module configuration XML

Now that we have set up our initial configuration, we need to set up our module-specific configuration which will specify the structural elements (models, classes, helpers, database tables and others) that the extension will use.

Our extension will add two pages to the frontend:

- One page that will contain a list of news posts
- A second page that show each individual news post.

The extension will also change the database to store data, and add the ability to manage news posts from the Magento Admin.

Based on these needs, we will need to define several items in our configuration file. We will revisit the different parts of this file when we discuss the implementation of the extension's features; for now, see the code snippet below for the configuration we will use.

config.xml

```
<?xml version="1.0"?>
<!--
/**
 * Module configuration
 *
 * @author Magento
 */
-->
<config>
    <modules>
        <Magentostudy_News>
            <version>1.0.0.0.1</version>
        </Magentostudy_News>
    </modules>
    <global>
        <models>
            <magentostudy_news>
                <class>Magentostudy_News_Model</class>
                <resourceModel>news_resource</resourceModel>
            </magentostudy_news>
            <news_resource>
                <class>Magentostudy_News_Model_Resource</class>
```

```
            <entities>
                <news>
                    <table>magentostudy_news</table>
                </news>
            </entities>
        </news_resource>
    </models>
    <helpers>
        <magentostudy_news>
            <class>Magentostudy_News_Helper</class>
        </magentostudy_news>
    </helpers>
    <blocks>
        <magentostudy_news>
            <class>Magentostudy_News_Block</class>
        </magentostudy_news>
    </blocks>
    <resources>
        <magentostudy_news_setup>
            <setup>
                <module>Magentostudy_News</module>
                <class>Mage_Core_Model_Resource_Setup</class>
            </setup>
        </magentostudy_news_setup>
    </resources>
    <events>
        <before_news_item_display>
            <observers>
                <magentostudy_news>
                    <class>magentostudy_news/observer</class>
                    <method>beforeNewsDisplayed</method>
                </magentostudy_news>
            </observers>
        </before_news_item_display>
    </events>
</global>
<frontend>
    <routers>
        <magentostudy_news>
            <use>standard</use>
            <args>
                <module>Magentostudy_News</module>
                <frontName>news</frontName>
            </args>
        </magentostudy_news>
    </routers>
    <layout>
        <updates>
            <magentostudy_news>
                <file>magentostudy_news.xml</file>
            </magentostudy_news>
        </updates>
    </layout>
</frontend>
<Admin>
    <routers>
        <adminhtml>
```

```
                <args>
                    <modules>
                        <Magentostudy_News
before="Mage_adminhtml">Magentostudy_News_adminhtml</Magentostudy_News>
                    </modules>
                </args>
            </adminhtml>
        </routers>
    </Admin>
    <adminhtml>
        <layout>
            <updates>
                <magentostudy_news>
                    <file>magentostudy_news.xml</file>
                </magentostudy_news>
            </updates>
        </layout>
    </adminhtml>
    <default>
        <news>
            <view>
                <enabled>1</enabled>
                <items_per_page>20</items_per_page>
                <days_difference>3</days_difference>
            </view>
        </news>
    </default>
</config>
```

Checking our module visibility

At this point, we have set up our initial and module-specific configuration settings. If we have done everything correctly, Magento should "see" our module. We can check this in the Admin panel.

Enter the Admin panel and navigate to System > Configuration > Advanced > Disable Modules Output to find our module named "Magentostudy_News".

Note
You may not see your module because of caching. There are two ways to resolve this:

- Clear the cache every time you make a change.
- Disable caching during the development process.

To clear the cache or change its settings, enter the Admin panel, navigate to System > Cache Management and use the "Flush Magento Cache" button.

To disable the cache, select all of the cache types, select the "Disable" from the dropdown, and press "Submit".

Getting Started: Design Setup

Now that we have got our code structure created and our extension configured, we will focus on creating our design structure. While Magento follows the basic ideas of the MVC design pattern, the "View" component is quite different from the Zend standard. Instead of a single file that receives data from models and logic from controllers, views in Magento are broken into Blocks, Layouts and Templates.

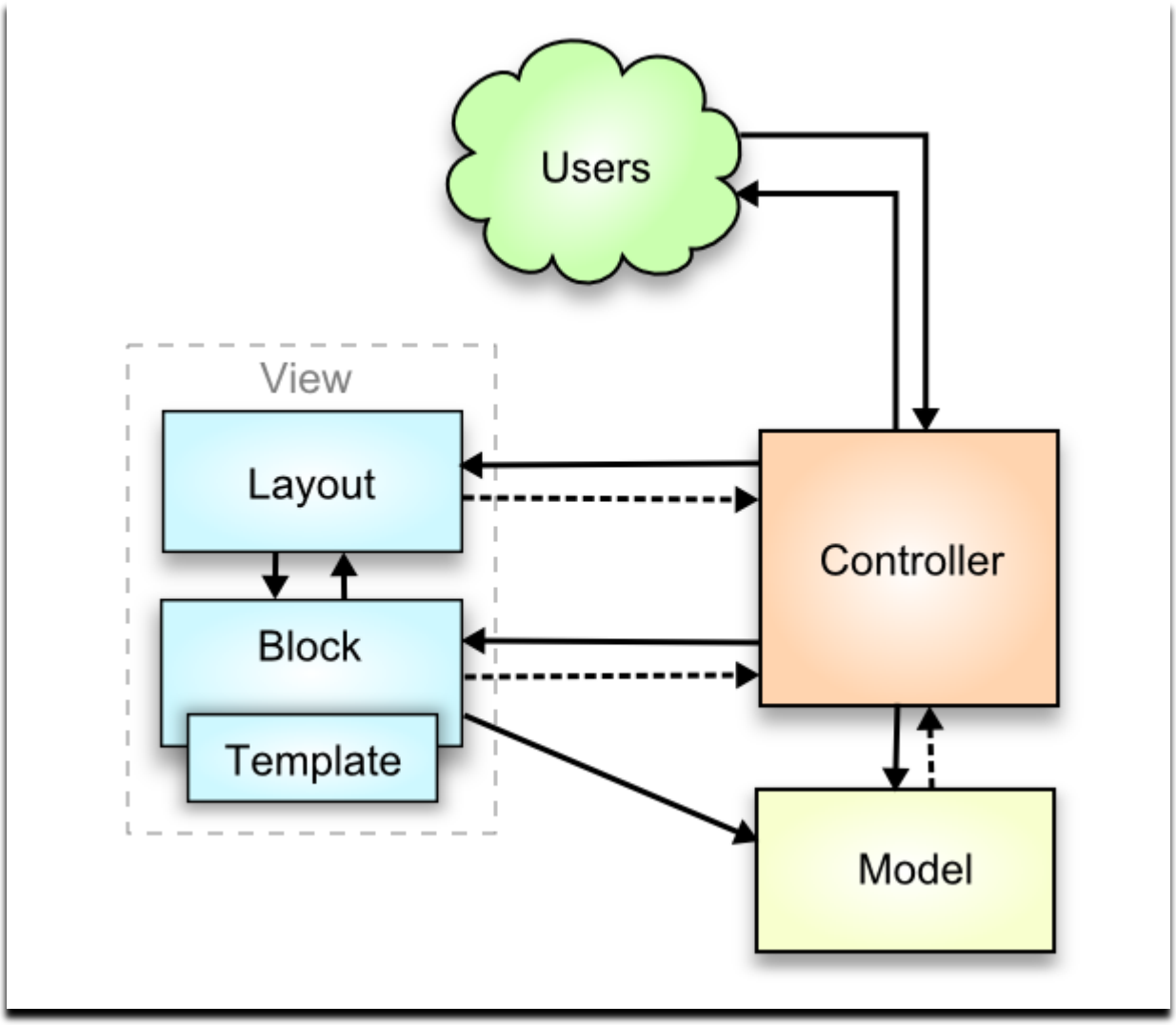

Figure 3: MVC design pattern for Magento

We have already prepared a structure for our blocks (See the `Block` folder in our code setup). Next we need to prepare the structure for our layouts and templates. We put them in the "design" folder. Because of the ability to use different themes in Magento and its fallback mechanism, all extension related theme files are placed in "`base/default`" package for frontend and "`default/default`" package for

`adminhtml`. This will allow extension to work correctly and independently on for theme chosen in Admin panel.

Extension Best Practices

Extension templates and layouts should be in stored in specific packages depending on whether they apply to the frontend or the Admin panel.

- Frontend templates and layouts should be stored in the `frontend/base/default` package.
- Admin panel templates and layouts should be stored in the: `adminhtml/default/default` package.
- In order to improve readability and make the relationship between templates and the module code more clear, templates should be stored in separate folders that are named with the module namespace and name. For example, in our extension we will create the folder structure for frontend templates so that the folder `magentostudy` will include the folder `news` where all our templates will be placed. And we place the newly created folder `magentostudy` into `frontend/base/default/template` according to the restrictions above.
- For similar reasons, layouts should be named with the module namespace and name. For example, in our extension we will name our frontend layout `magentostudy{{_news.xml}}` and place it into `frontend/base/default/layout`.
- Be advised that native layouts don't include a namespace in their names. For example, the layout related to the core module `Mage_Catalog` is named just `catalog.xml`, not `mage_catalog.xml`

Creating our template and layout structures

Because our module will affect both the Admin panel and the frontend, we need to create file/folder structures for both. As with our code structure mentioned earlier, you should create only those folders that are necessary. For example, if you have a layout for the backend, but no templates for it, you shouldn't create a template folder for that layout. See the following diagrams for the frontend and Admin panel structures:

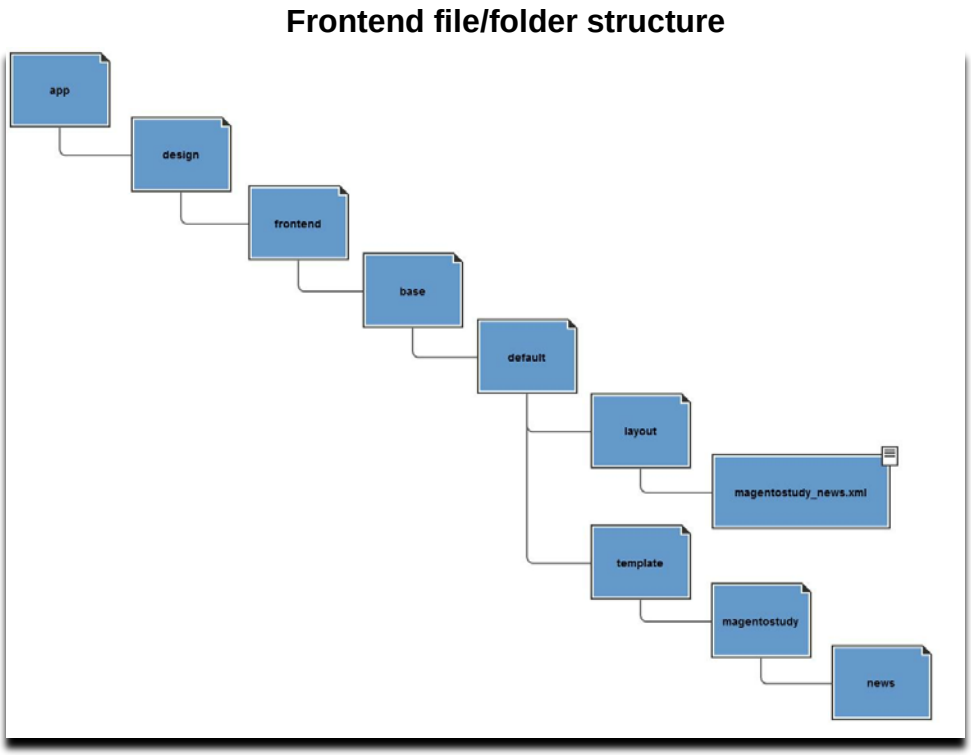

Figure 4: Frontend file/folder structure

Admin panel file/folder structure

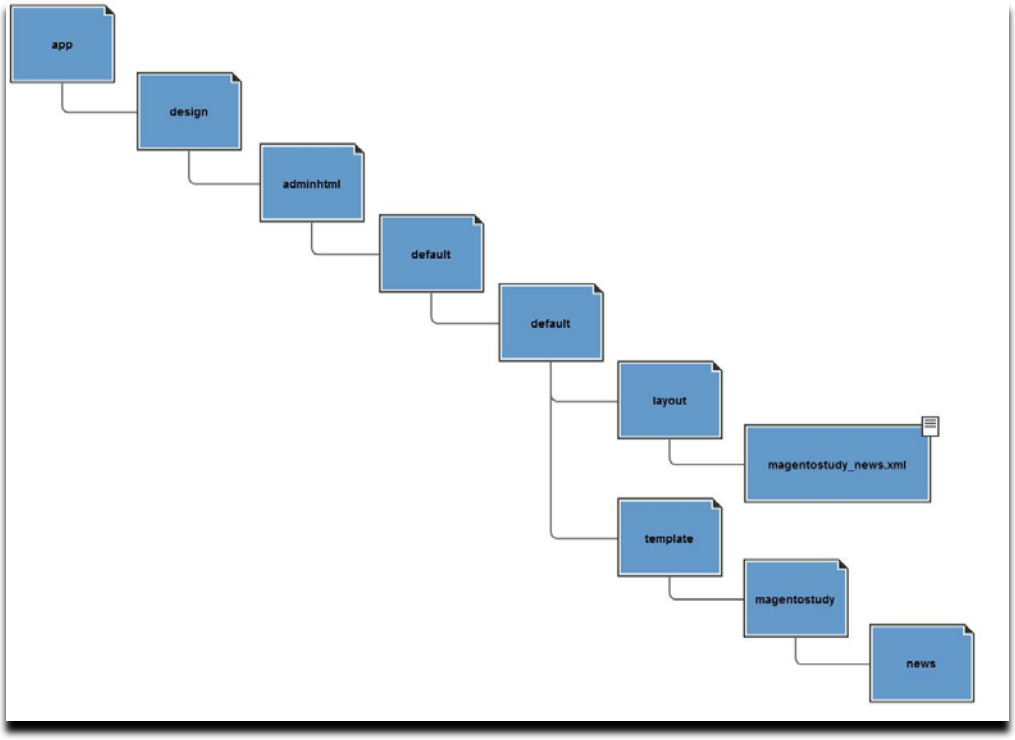

Figure 5: Admin panel file/folder structure

Auxiliary files

Along with templates and layouts, an extension might also need some additional files such as images, JavaScript, CSS files, or other external libraries.

JavaScript (JS), Cascading Style Sheets (CSS)

JavaScript (JS)

There are two locations you can use to store JavaScript files -- in the application js folder or in an appropriate skin folder. The application js folder should be used if the provided code may be helpful for another extension, for example, some library. If it is an extension-specific JavaScript it should be placed in the skin folder.

Extension Best Practices

There are two locations you can use to store JavaScript files -- in the application `js` folder or an appropriate `skin` folder.

- If you use the global `js` folder, your JavaScript will need to be stored in a second nested folder that is named according to the module or library to which the `js` applies. For example, any JavaScript for our extension would be stored in: `js/magentostudy_news`. Similarly, any JavaScript that customizes an external library would be stored in `js/<library name>`.

- Using the global approach, you can store JavaScript for the Admin panel in the same js folder for your module (`js/magentostudy_news`).
- If you use the `skin` folder, your frontend JavaScript should be stored in
- `skin/frontend/base/default/js/<namespace>_<name>`. In our case, `skin/frontend/base/default/js/magentostudy_news`
- Using the skin approach, Admin panel JavaScript should be stored in `skin/adminhtml/default/default/js/<namespace>_<name>`. In our case, `skin/adminhtml/default/default/js/magentostudy_news`

Extension-specific JavaScript files should not overwrite any native file.

Cascading Style Sheets (CSS)

Extension Best Practices

Extension-specific CSS files should be stored in `css` folders within an appropriate `skin` folder depending on whether they apply to the frontend or the Admin panel.

- Frontend css: `skin/frontend/base/default/css`. In our case, `skin/frontend/base/default/css/magentostudy_news.css`
- Admin panel css: `skin/adminhtml/default/default/css`. In our case, `skin/adminhtml/default/default/css/magentostudy_news.css`

Extension-specific CSS files should not overwrite any native files.

Media and External Libraries

Extension Best Practices

- Images and other media files should be stored in the `media` folder
- External libraries needed by an extension should be stored in the `lib` folder
- External libraries should be open source.

Development: The Front End

Now that we have created our module's structure and configured it, we can begin our frontend development. As previously mentioned, our extension will consist of two frontend pages. One will list all of our news posts; the second will display the content of individual posts from the list.

Let's focus on our list of posts and get that to show up on the frontend. Our list of news posts will be accessible via a footer link or a direct URL. Here's what we will need to do:

- Add tables to our database schema to store our news posts
- Prepare models to interact with the database and the news post data

- Specify routes between our pages
- Prepare output for our pages
- Prepare auxiliary helpers for additional data manipulation
- Add some sample data and test our results

Preparing our database schema

Starting from Magento Enterprise Edition (EE) 1.11 and Magento Community Edition (CE) 1.6 you can use "data" folder for install/upgrade scripts which manipulate data and "`sql`" folder for scripts that alter database schema. This feature allows to separate database structure from data.

Magento does not work directly with raw SQL. Instead, it uses aliases that refer to the SQL itself. For example, the core database table `sales_flat_order_item` is referred to with the alias `sales/order_item`. Before we add any new tables or alter existing ones in our schema, we need to configure their aliases in the module config xml. Our extension will use a table named `magentostudy_news`.

Extension Best Practices

- Name database tables by module namespace and name to make the relationship between the code and the database clearer.

config.xml specifying module's database table

```
<entities>
    <news>
        <table>magentostudy_news</table>
    </news>
</entities>
```

Configuring our database schema script

Magento alters the database schema using PHP scripts that modify the SQL structure using special adapter methods. These scripts are configured within the `<resources>` node in our `config.xml` file. Each script is stored in a nested folder that is referred to by the first child in the `<resources>` node (in our case, `magentostudy_news_setup`. A third node, `<setup>`, defines which class our install script will use for our extension. Different modules might require extended setup classes -- for example, the sales setup class will automatically add sales aliases for database tables related to sales so you don't need to add them manually. For our extension, since we are only adding a new table, we don't need to do anything specific in the setup class. We will use the native core class.

config.xml (specifying setup class)

```
<resources>
    <magentostudy_news_setup>
        <setup>
            <module>Magentostudy_News</module>
            <class>Mage_Core_Model_Resource_Setup</class>
        </setup>
    </magentostudy_news_setup>
</resources>
```

Writing our schema install script

Next, we need to write our schema installation PHP script. Before we start, we need to decide whcih data we will store in the database. For our extension, we will store the following:

- Title
- Author
- Content
- Image
- Date when news was published
- Date when news was created on Magento instance
- News unique identifier

Our list of news posts will be ordered by the `published_at` field, so to increase performance, we will create a database index on this field.

Before looking at our database alteration snippet, here are some restrictions and recommendations that you should observe when working with databases and data manipulation:

Extension Best Practices

- Do not create references from core database tables to extension ones, because this could break native Magento functionality.
- Do not alter database structure on the fly. All changes should be applied via database install or upgrade scripts.
- Use database indexes when creating tables, because this can significantly improve database (and overall) performance. More than one index can be created for a table.
- Data and structure upgrades should be processed via Adapter methods; raw SQL is not recommended. Usage of adapter methods will make an extension independent of concrete database realization.
- If your extension stores some data such as statistics, the best approach is to create necessary logic using database storage instead of the file system.

Separate data upgrades from schema upgrades. Pay strict attention to this because different upgrades were added in Magento Enterprise Edition (EE) 1.11 and Magento Community Edition (CE) 1.6.

install-1.0.0.0.1.php

```
<?php
/**
 * News installation script
 *
 * @author Magento
 */

/**
 * @var $installer Mage_Core_Model_Resource_Setup
 */
$installer = $this;

/**
 * Creating table magentostudy_news
 */
$table = $installer->getConnection()
    ->newTable($installer->getTable('magentostudy_news/news'))
    ->addColumn('news_id', Varien_Db_Ddl_Table::TYPE_INTEGER, null, array(
        'unsigned' => true,
        'identity' => true,
        'nullable' => false,
        'primary'  => true,
    ), 'Entity id')
    ->addColumn('title', Varien_Db_Ddl_Table::TYPE_TEXT, 255, array(
        'nullable' => true,
    ), 'Title')
    ->addColumn('author', Varien_Db_Ddl_Table::TYPE_TEXT, 63, array(
        'nullable' => true,
        'default'  => null,
    ), 'Author')
    ->addColumn('content', Varien_Db_Ddl_Table::TYPE_TEXT, '2M', array(
        'nullable' => true,
        'default'  => null,
    ), 'Content')
    ->addColumn('image', Varien_Db_Ddl_Table::TYPE_TEXT, null, array(
        'nullable' => true,
        'default'  => null,
    ), 'News image media path')
    ->addColumn('published_at', Varien_Db_Ddl_Table::TYPE_DATE, null, array(
        'nullable' => true,
        'default'  => null,
    ), 'World publish date')
    ->addColumn('created_at', Varien_Db_Ddl_Table::TYPE_TIMESTAMP, null,
array(
        'nullable' => true,
        'default'  => null,
    ), 'Creation Time')
    ->addIndex($installer->getIdxName(
            $installer->getTable('magentostudy_news/news'),
            array('published_at'),
            Varien_Db_Adapter_Interface::INDEX_TYPE_INDEX
        ),
        array('published_at'),
        array('type' => Varien_Db_Adapter_Interface::INDEX_TYPE_INDEX)
```

```
    )
    ->setComment('News item');

$installer->getConnection()->createTable($table);
```

Preparing our models to work with data and business logic

There are two model types in Magento:

- **Domain Models** are responsible for implementing business logic and manipulating combinations of data from resource models, APIs and other sources before the data is inserted or updated in the database. If you are working with databases, domain models depend on resource models.
- **Resource Models** perform low-level SQL actions. They act as an abstraction layer between domain models and the database.

Defining our models

Before implementing our models, we need to define them in `config.xml`. Our model and resource model use classes whose names begin with the value in the `<class>` node. (This value is not an actual model name, but rather the prefix used to specify our models). Similarly, our resource models use a prefix of `Magentostudy_News_Model_Resource`.

config.xml (models section)

```
<models>
    <magentostudy_news>
        <class>Magentostudy_News_Model</class>
        <resourceModel>news_resource</resourceModel>
    </magentostudy_news>
    <news_resource>
        <class>Magentostudy_News_Model_Resource</class>
        <entities>
            <news>
                <table>magentostudy_news</table>
            </news>
        </entities>
    </news_resource>
</models>
```

Implementing our model

Extension Best Practices

- Data should be escaped before inserting it into the database. This is extremely important for security because it helps to avoid SQL-injections.
- The extension should be responsible for validating incoming data such as user entries, API call responses, etc.

Magento's abstract core model already implements basic methods. By extending our model from this one, we'll be able to use basic methods such as load, save, and delete. We will use our model to update the created field if the post is new. For this, we need to create class `Magentostudy/News/Model/News.php`.

Magentostudy_News_Model_News

```
<?php
/**
 * News item model
 *
 * @author Magento
 */
class Magentostudy_News_Model_News extends Mage_Core_Model_Abstract
{
    /**
     * Define resource model
     */
    protected function _construct()
    {
        $this->_init('magentostudy_news/news');
    }

    /**
     * If object is new adds creation date
     *
     * @return Magentostudy_News_Model_News
     */
    protected function _beforeSave()
    {
        parent::_beforeSave();
        if ($this->isObjectNew()) {
            $this->setData('created_at', Varien_Date::now());
        }
        return $this;
    }
}
```

Implementing our resource model and collection

Extension Best Practices

- Be sure to use the correct connection for queries. If you want only select records – use read connection, otherwise use write connection. Pay attention that all queries like "SELECT … FOR UPDATE" also need write permission, so write connection should be used. Pay attention to queries combined into transactions – if only one query needs write permissions, all queries in the transaction should use write connection
- Database query manipulations should be placed in resource models only. Resource models are the best place for SQL.
- Use Zend style SQL queries because they make SQL queries more readable and provides object-oriented approach, which allows modifying them in other modules.

Our plan is to load, save, or delete individual posts and to display a collection of posts on the frontend. As with our domain models, we do not need to implement basic

actions; we only need to specify the resource we want to work with to return its data. In our collection, we will implement a method for sorting news posts and limiting their count.

We will add two files: `Magentostudy/News/Model/Resource/News.php` for our individual posts and `Magentostudy/News/Model/Resource/News/Collection.php` for our collection of all posts.

Magentostudy_News_Model_Resource_News

```
<?php
/**
 * News item resource model
 *
 * @author Magento
 */
class Magentostudy_News_Model_Resource_News extends
Mage_Core_Model_Resource_Db_Abstract
{
    /**
     * Initialize connection and define main table and primary key
     */
    protected function _construct()
    {
        $this->_init('magentostudy_news/news', 'news_id');
    }
}
```

Magentostudy_News_Model_Resource_News_Collection

```
<?php
/**
 * News collection
 *
 * @author Magento
 */
class Magentostudy_News_Model_Resource_News_Collection extends
Mage_Core_Model_Resource_Db_Collection_Abstract
{
    /**
     * Define collection model
     */
    protected function _construct()
    {
        $this->_init('magentostudy_news/news');
    }

    /**
     * Prepare for displaying in list
     *
     * @param integer $page
     * @return Magentostudy_News_Model_Resource_News_Collection
     */
    public function prepareForList($page)
```



```
    {
        $this->setPageSize(Mage::helper('magentostudy_news')-
>getNewsPerPage());
        $this->setCurPage($page)->setOrder('published_at',
Varien_Data_Collection::SORT_ORDER_DESC);
        return $this;
    }
}
```

Specifying our routes and request flow

In MVC architecture, controllers are responsible for routes and request flow. Our news extension will use a controller that will make our pages accessible by URLs on the frontend.

Configuring our routes

Before implementing our controller, we need to configure our routes in `config.xml`.

Here, the `frontName` node specifies that our controller will be accessible through the frontend URL `http://<store-URL>/index.php/news`.

config.xml (specifying routes)

```
<frontend>
    <routers>
        <magentostudy_news>
            <use>standard</use>
            <args>
                <module>Magentostudy_News</module>
                <frontName>news</frontName>
            </args>
        </magentostudy_news>
    </routers>
```

Implementing our controller

According to our plan, our controller needs two actions: a news list and a single post view. `indexAction` is default controller action and will be responsible for the news list. The action `viewAction` will be responsible for the individual news posts. Before implementing our controller, let's consider some restrictions and recommendations for writing controllers in Magento extensions:

Extension Best Practices

- Do not use PHP global variables like `$_SERVER, $_GET, $_POST`. There are special Magento methods to interact with this data that should be used instead. For example, you can use the `getRequest()` function in a controller method to get request data.
- Do not work directly with the PHP `$_GLOBAL` variable – Magento provides an alternative solution to avoid usage of `$_GLOBAL` - `Mage::register()`, `Mage::registry()`, `Mage::unregister()`.
- Override controllers only if you do not have another more flexible solution.
- If you plan to just add new action(s) without using methods or variables from the parent, use an abstract class for the parent. This will help keep your extension compatible with others.
- According to the Zend Framework MVC best practices, the controller component of an application "must be as skinny as possible". It is strongly recommended to encapsulate all business logic into models and helpers; avoid implementing business logic in a controller action.

Now, let's add our controller
`Magentostudy/News/controller/IndexController.php`

Magentostudy_News_IndexController

```
<?php
 /**
 * News frontend controller
 *
 * @author Magento
 */
class Magentostudy_News_IndexController extends
Mage_Core_Controller_Front_Action
{
    /**
     * Pre dispatch action that allows to redirect to no route page in case
of disabled extension through Admin panel
     */
    public function preDispatch()
    {
        parent::preDispatch();

        if (!Mage::helper('magentostudy_news')->isEnabled()) {
            $this->setFlag('', 'no-dispatch', true);
            $this->_redirect('noRoute');
        }
    }

    /**
     * Index action
     */
    public function indexAction()
    {
        $this->loadLayout();

        $listBlock = $this->getLayout()->getBlock('news.list');

        if ($listBlock) {
            $currentPage = abs(intval($this->getRequest()->getParam('p')));
```

```
            if ($currentPage < 1) {
                $currentPage = 1;
            }
            $listBlock->setCurrentPage($currentPage);
        }

        $this->renderLayout();
    }

    /**
     * News view action
     */
    public function viewAction()
    {
        $newsId = $this->getRequest()->getParam('id');
        if (!$newsId) {
            return $this->_forward('noRoute');
        }

        /** @var $model Magentostudy_News_Model_News */
        $model = Mage::getModel('magentostudy_news/news');
        $model->load($newsId);

        if (!$model->getId()) {
            return $this->_forward('noRoute');
        }

        Mage::register('news_item', $model);

        Mage::dispatchEvent('before_news_item_display', array('news_item' =>
$model));

        $this->loadLayout();
        $itemBlock = $this->getLayout()->getBlock('news.item');
        if ($itemBlock) {
            $listBlock = $this->getLayout()->getBlock('news.list');
            if ($listBlock) {
                $page = (int)$listBlock->getCurrentPage() ? (int)$listBlock-
>getCurrentPage() : 1;
            } else {
                $page = 1;
            }
            $itemBlock->setPage($page);
        }
        $this->renderLayout();
    }
}
```

Preparing our output

Configuring our output

Before implementing output components, will need to configure our layouts and blocks in our `config.xml` file.

config.xml (specifying frontend layout)

```
<layout>
    <updates>
        <magentostudy_news>
            <file>magentostudy_news.xml</file>
        </magentostudy_news>
    </updates>
</layout>
```

config.xml (specifying blocks)

```
<blocks>
    <magentostudy_news>
        <class>Magentostudy_News_Block</class>
    </magentostudy_news>
</blocks>
```

Implementing our layout

As shown above, each of our controller's actions loads and renders a layout. This layout does not exist yet, so next we will create it.

Extension Best Practice

- It's not recommended to remove native blocks in custom layouts. Add new blocks instead.

Magentostudy_news.xml

```
<?xml version="1.0"?>
<!--
/**
 * Magento frontend layout
 *
 * @author Magento
 */
-->
<layout version="0.1.0">
    <default>
        <reference name="footer_links">
            <action method="addLink" translate="label title"
module="magentostudy_news" ifconfig="news/view/enabled">
                <label>News</label>
                <url>news</url>
                <title>News</title>
                <prepare>true</prepare>
            </action>
        </reference>
    </default>

    <magentostudy_news_index_index translate="label">
        <label>News Page</label>
```

```
        <reference name="root">
            <action method="setTemplate">
                <template>page/2columns-right.phtml</template>
            </action>
            <action method="setHeaderTitle" translate="title"
module="magentostudy_news">
                <title>Site News</title>
            </action>
        </reference>
        <reference name="content">
            <block type="magentostudy_news/list" name="news.list"
template="magentostudy/news/list.phtml">
                <block type="page/html_pager" name="news.list.pager"
as="news_list_pager" />
            </block>
        </reference>
    </magentostudy_news_index_index>

    <magentostudy_news_index_view translate="label">
        <label>News Item Page</label>
        <reference name="root">
            <action method="setTemplate">
                <template>page/2columns-right.phtml</template>
            </action>
        </reference>
        <reference name="content">
            <block type="magentostudy_news/news" name="news.item"
template="magentostudy/news/item.phtml" />
        </reference>
    </magentostudy_news_index_view>
</layout>
```

Implementing our blocks

In our layout above, we add three view blocks: one for our footer link, a second for our list of news posts and, a third for our individual posts. We do not need to add anything more for our footer link block because it is a native Magento block and we are simply adding our link to it. However, we will need to create classes for our list and item blocks, `Magentostudy/News/Block/List.php` and `Magentostudy/News/Block/Item.php`.

Magentostudy_News_Block_List

```
<?php
/**
 * News List block
 *
 * @author Magento
 */
class Magentostudy_News_Block_List extends Mage_Core_Block_Template
{
    /**
     * News collection
     *
```

```
     * @var Magentostudy_News_Model_Resource_News_Collection
     */
    protected $_newsCollection = null;

    /**
     * Retrieve news collection
     *
     * @return Magentostudy_News_Model_Resource_News_Collection
     */
    protected function _getCollection()
    {
        return  Mage::getResourceModel('magentostudy_news/news_collection');
    }

    /**
     * Retrieve prepared news collection
     *
     * @return Magentostudy_News_Model_Resource_News_Collection
     */
    public function getCollection()
    {
        if (is_null($this->_newsCollection)) {
            $this->_newsCollection = $this->_getCollection();
            $this->_newsCollection->prepareForList($this->getCurrentPage());
        }

        return $this->_newsCollection;
    }

    /**
     * Return URL to item's view page
     *
     * @param Magentostudy_News_Model_News $newsItem
     * @return string
     */
    public function getItemUrl($newsItem)
    {
        return $this->getUrl('*/*/view', array('id' => $newsItem->getId()));
    }

    /**
     * Fetch the current page for the news list
     *
     * @return int
     */
    public function getCurrentPage()
    {
        return $this->getData('current_page') ? $this-
>getData('current_page') : 1;
    }

    /**
     * Get a pager
     *
     * @return string|null
     */
    public function getPager()
```



```
    {
        $pager = $this->getChild('news_list_pager');
        if ($pager) {
            $newsPerPage = Mage::helper('magentostudy_news')-
>getNewsPerPage();

            $pager->setAvailableLimit(array($newsPerPage => $newsPerPage));
            $pager->setTotalNum($this->getCollection()->getSize());
            $pager->setCollection($this->getCollection());
            $pager->setShowPerPage(true);

            return $pager->toHtml();
        }

        return null;
    }

    /**
     * Return URL for resized News Item image
     *
     * @param Magentostudy_News_Model_News $item
     * @param integer $width
     * @return string|false
     */
    public function getImageUrl($item, $width)
    {
        return Mage::helper('magentostudy_news/image')->resize($item,
$width);
    }
}
```

Magentostudy_News_Block_News

```
<?php
/**
 * News Item block
 *
 * @author Magento
 */
class Magentostudy_News_Block_Item extends Mage_Core_Block_Template
{
    /**
     * Current news item instance
     *
     * @var Magentostudy_News_Model_News
     */
    protected $_item;

    /**
     * Return parameters for back url
     *
     * @param array $additionalParams
     * @return array
     */
    protected function _getBackUrlQueryParams($additionalParams = array())
    {
```

```
        return array_merge(array('p' => $this->getPage()),
$additionalParams);
    }

    /**
     * Return URL to the news list page
     *
     * @return string
     */
    public function getBackUrl()
    {
        return $this->getUrl('*/', array('_query' => $this-
>_getBackUrlQueryParams()));
    }

    /**
     * Return URL for resized News Item image
     *
     * @param Magentostudy_News_Model_News $item
     * @param integer $width
     * @return string|false
     */
    public function getImageUrl($item, $width)
    {
        return Mage::helper('magentostudy_news/image')->resize($item,
$width);
    }
}
```

These blocks illustrate some Magento best practices and restrictions which you should consider when developing your extensions:

Extension Best Practices

- Use special Magento methods for URL generation such as `Mage::getURL()`. Using these methods allows us to generate URLs based on different criteria such as security options.
- Line 51 in `Magentostudy_News_Block_List` shows an example of how to use `getURL()` to generate a URL for the individual news view.
-
- Frontend blocks should be cached, wherever possible to optimize frontend performance.
- All blocks except special dynamic content should be cached using native Magento cache functionality - methods `Mage::app()->saveCache()`and `Mage::app()->` loadCache().
- Use logical caching. For example, store values that don't change during calls to different class methods in protected class variables and use these values instead of loading new values or making new calculations.

In our case, we have a good example of logical caching. Take a look at block `Magentostudy_News_Block_List`. Here we manipulate a collection of news, but in the `getCollection` method, we do not instantiate and apply filters to the collection every time. We can do this one time and then save the collection into a protected variable - `$_newsCollection` -- for further use.

Implementing our templates

Our templates are already configured in our layout files so we will start implementing them. Before we begin, let's consider some restrictions and recommendations for writing templates in Magento extensions:

Extension Best Practices

- Core templates files cannot be physically overridden in the extension package. Use layout functionality to extend frontend views.
- All phrases must be enclosed with translation tags and methods so that they can be translated according to the store locale. (This applies to all output, including phrases that appear in controllers, blocks and helpers.)
- All data shown in templates should be preliminarily escaped with the `escapeHtml`, `stripTags` methods to avoid possible XSS injections.
- Templates shouldn't contain business logic. Move all business logic to models, blocks, and helpers.

In our layout we've specified two templates - `magentostudy/news/list.phtml` and `magentostudy/news/item.phtml` for list of news and separate news post respectively.

list.phtml

```
<?php
/**
 * News template for items list
 *
 * @author Magento
 */

/**
 * @var $this Magentostudy_News_Block_List
 * @see Magentostudy_News_Block_List
 */
?>
<div id="news_list_messages"><?php echo $this->getMessagesBlock()-
>getGroupedHtml() ?></div>
<div>
    <h1><?php echo Mage::helper('magentostudy_news')->__('Site News') ?></h1>
</div>

<div>
    <?php foreach ($this->getCollection() as $newsItem): ?>
        <div id="item_<?php echo $newsItem->getId() ?>">
            <h2>
                <a href="<?php echo $this->getItemUrl($newsItem) ?>">
                    <?php echo $this->escapeHtml($newsItem->getTitle()) ?>
                </a>
            </h2>

            <div>
```

```
                <?php echo $this->formatDate($newsItem->getTimePublished(),
'medium') ?> |
                <?php echo $this->escapeHtml($newsItem->getAuthor()) ?>
            </div>

            <?php if ($imageUrl = $this->getImageUrl($newsItem, 100)): ?>
               <p><img src="<?php echo $imageUrl ?>" alt="<?php echo $this-
>escapeHtml($newsItem->getTitle()); ?>" /></p>
            <?php endif; ?>
        </div>
    <?php endforeach; ?>
</div>

<?php echo $this->getPager() ?>
```

item.phtml

```
<?php
/**
 * News template for separate item
 *
 * @author Magento
 */

/**
 * @var $this Magentostudy_News_Block_Item
 * @see Magentostudy_News_Block_Item
 */
?>
<?php $_newsItem = $this->helper('magentostudy_news')->getNewsItemInstance();
?>
<div id="news_item_messages"><?php echo $this->getMessagesBlock()-
>getGroupedHtml() ?></div>

<div>
    <h1><?php echo $this->escapeHtml($_newsItem->getTitle()) ?></h1>

    <div>
        <?php echo $this->formatDate($_newsItem->getTimePublished(),
Mage_Core_Model_Locale::FORMAT_TYPE_MEDIUM) ?> |
        <?php echo $this->escapeHtml($_newsItem->getAuthor()) ?>
    </div>
</div>

<div>
    <?php if ($imageUrl = $this->getImageUrl($_newsItem, 400)): ?>
       <p><img src="<?php echo $imageUrl ?>" alt="<?php echo $this-
>escapeHtml($_newsItem->getTitle()) ?>" /></p>
    <?php endif; ?>

    <div><?php echo $_newsItem->getContent() ?></div>

    <div>
        <a href="<?php echo $this->getBackUrl() ?>">
            <?php echo Mage::helper('magentostudy_news')->__('Return to
list') ?>
        </a>
```

```
    </div>
</div>
```

Preparing our helpers

As you may have seen in previous code snippets there are many helper calls. Helpers are auxiliary classes that implement commonly used business logic.

In our extension we will have two frontend helpers:

- A **data helper** used for translation and other interactions with the store's configuration
- An **image helper** that provides functionality to work with images (uploading, resizing, removing, etc.)

Extension Best Practices

- An extension should have its own helper for translations, to allow different phrase translations in different modules and to make the module independent from others.
- Do not hardcode pre-defined values directly into methods. Use class constants, protected class variables, or store configuration to allow for changing values without code changes.

In our case, we will use store configuration to allow certain values to be changed. We won't hard-code the number of posts per page or the number of days a post should be considered recent.

Configuring our helpers

Before implementing our helpers, we need toconfigure them in our `config.xml` file.

config.xml (specifying helpers)

```
<helpers>
    <magentostudy_news>
        <class>Magentostudy_News_Helper</class>
    </magentostudy_news>
</helpers>
```

Implementing our helpers

Now we'll add our two helpers:

- `Magentostudy/News/Helper/Data.php`
- `Magentostudy/News/Helper/Image.php`

Magentostudy_News_Helper_Data

```
<?php
/**
```

```
 * News Data helper
 *
 * @author Magento
 */
class Magentostudy_News_Helper_Data extends Mage_Core_Helper_Data
{
    /**
     * Path to store config if frontend output is enabled
     *
     * @var string
     */
    const XML_PATH_ENABLED            = 'news/view/enabled';

    /**
     * Path to store config where count of news posts per page is stored
     *
     * @var string
     */
    const XML_PATH_ITEMS_PER_PAGE     = 'news/view/items_per_page';

    /**
     * Path to store config where count of days while news is still recently
added is stored
     *
     * @var string
     */
    const XML_PATH_DAYS_DIFFERENCE    = 'news/view/days_difference';

    /**
     * News Item instance for lazy loading
     *
     * @var Magentostudy_News_Model_News
     */
    protected $_newsItemInstance;

    /**
     * Checks whether news can be displayed in the frontend
     *
     * @param integer|string|Mage_Core_Model_Store $store
     * @return boolean
     */
    public function isEnabled($store = null)
    {
        return Mage::getStoreConfigFlag(self::XML_PATH_ENABLED, $store);
    }

    /**
     * Return the number of items per page
     *
     * @param integer|string|Mage_Core_Model_Store $store
     * @return int
     */
    public function getNewsPerPage($store = null)
    {
        return abs((int)Mage::getStoreConfig(self::XML_PATH_ITEMS_PER_PAGE,
$store));
    }
```



```
    /**
     * Return difference in days while news is recently added
     *
     * @param integer|string|Mage_Core_Model_Store $store
     * @return int
     */
    public function getDaysDifference($store = null)
    {
        return abs((int)Mage::getStoreConfig(self::XML_PATH_DAYS_DIFFERENCE,
$store));
    }

    /**
     * Return current news item instance from the Registry
     *
     * @return Magentostudy_News_Model_News
     */
    public function getNewsItemInstance()
    {
        if (!$this->_newsItemInstance) {
            $this->_newsItemInstance = Mage::registry('news_item');

            if (!$this->_newsItemInstance) {
                Mage::throwException($this->__('News item instance does not
exist in Registry'));
            }
        }

        return $this->_newsItemInstance;
    }
}
```

Our mage helper works with physical files, so take a look at Magento registered methods that avoid direct work with physical files paths:

Extension Best Practices

- If you are working with physical files, use Magento's registered methods (for example, method `Mage::getBaseDir()`) to get links to them.

Magentostudy_News_Helper_Image

```
<?php
/**
 * News Image Helper
 *
 * @author Magento
 */
class Magentostudy_News_Helper_Image extends Mage_Core_Helper_Abstract
{
    /**
```

```
     * Media path to extension imahes
     *
     * @var string
     */
    const MEDIA_PATH    = 'news';

    /**
     * Maximum size for image in bytes
     * Default value is 1M
     *
     * @var int
     */
    const MAX_FILE_SIZE = 1048576;

    /**
     * Manimum image height in pixels
     *
     * @var int
     */
    const MIN_HEIGHT = 50;

    /**
     * Maximum image height in pixels
     *
     * @var int
     */
    const MAX_HEIGHT = 800;

    /**
     * Manimum image width in pixels
     *
     * @var int
     */
    const MIN_WIDTH = 50;

    /**
     * Maximum image width in pixels
     *
     * @var int
     */
    const MAX_WIDTH = 800;

    /**
     * Array of image size limitation
     *
     * @var array
     */
    protected $_imageSize   = array(
        'minheight'     => self::MIN_HEIGHT,
        'minwidth'      => self::MIN_WIDTH,
        'maxheight'     => self::MAX_HEIGHT,
        'maxwidth'      => self::MAX_WIDTH,
    );

    /**
     * Array of allowed file extensions
     *
```

```
     * @var array
     */
    protected $_allowedExtensions = array('jpg', 'gif', 'png');

    /**
     * Return the base media directory for News Item images
     *
     * @return string
     */
    public function getBaseDir()
    {
        return Mage::getBaseDir('media') . DS . self::MEDIA_PATH;
    }

    /**
     * Return the Base URL for News Item images
     *
     * @return string
     */
    public function getBaseUrl()
    {
        return Mage::getBaseUrl('media') . '/' . self::MEDIA_PATH;
    }

    /**
     * Remove news item image by image filename
     *
     * @param string $imageFile
     * @return bool
     */
    public function removeImage($imageFile)
    {
        $io = new Varien_Io_File();
        $io->open(array('path' => $this->getBaseDir()));
        if ($io->fileExists($imageFile)) {
            return $io->rm($imageFile);
        }
        return false;
    }

    /**
     * Upload image and return uploaded image file name or false
     *
     * @throws Mage_Core_Exception
     * @param string $scope the request key for file
     * @return bool|string
     */
    public function uploadImage($scope)
    {
        $adapter  = new Zend_File_Transfer_Adapter_Http();
        $adapter->addValidator('ImageSize', true, $this->_imageSize);
        $adapter->addValidator('Size', true, self::MAX_FILE_SIZE);
        if ($adapter->isUploaded($scope)) {
            // validate image
            if (!$adapter->isValid($scope)) {
                Mage::throwException(Mage::helper('magentostudy_news')-
>__('Uploaded image is not valid'));
```

```
            }
            $upload = new Varien_File_Uploader($scope);
            $upload->setAllowCreateFolders(true);
            $upload->setAllowedExtensions($this->_allowedExtensions);
            $upload->setAllowRenameFiles(true);
            $upload->setFilesDispersion(false);
            if ($upload->save($this->getBaseDir())) {
                return $upload->getUploadedFileName();
            }
        }
        return false;
    }

    /**
     * Return URL for resized News Item Image
     *
     * @param Magentostudy_News_Model_Item $item
     * @param integer $width
     * @param integer $height
     * @return bool|string
     */
    public function resize(Magentostudy_News_Model_News $item, $width,
$height = null)
    {
        if (!$item->getImage()) {
            return false;
        }

        if ($width < self::MIN_WIDTH || $width > self::MAX_WIDTH) {
            return false;
        }
        $width = (int)$width;

        if (!is_null($height)) {
            if ($height < self::MIN_HEIGHT || $height > self::MAX_HEIGHT) {
                return false;
            }
            $height = (int)$height;
        }

        $imageFile = $item->getImage();
        $cacheDir  = $this->getBaseDir() . DS . 'cache' . DS . $width;
        $cacheUrl  = $this->getBaseUrl() . '/' . 'cache' . '/' . $width .
'/';

        $io = new Varien_Io_File();
        $io->checkAndCreateFolder($cacheDir);
        $io->open(array('path' => $cacheDir));
        if ($io->fileExists($imageFile)) {
            return $cacheUrl . $imageFile;
        }

        try {
            $image = new Varien_Image($this->getBaseDir() . DS . $imageFile);
            $image->resize($width, $height);
            $image->save($cacheDir . DS . $imageFile);
            return $cacheUrl . $imageFile;
```

```
        } catch (Exception $e) {
            Mage::logException($e);
            return false;
        }
    }

    /**
     * Removes folder with cached images
     *
     * @return boolean
     */
    public function flushImagesCache()
    {
        $cacheDir  = $this->getBaseDir() . DS . 'cache' . DS ;
        $io = new Varien_Io_File();
        if ($io->fileExists($cacheDir, false) ) {
            return $io->rmdir($cacheDir, true);
        }
        return true;
    }
}
```

Adding sample data

Now, we are ready to view our extension on the frontend. Before we can see anything, we will need to add some sample data to the database. We will use a data install script for this: `data/magentostudy_news_setup/data-install-1.0.0.0.1.php`. We do not need to add anything to `config.xml` because we have already specified everything we needed when we configured our database schema installation scripts earlier.

data-install-1.0.0.0.1.php

```
<?php
/**
 * News data installation script
 *
 * @author Magento
 */

/**
 *  @var $installer Mage_Core_Model_Resource_Setup
 */
$installer = $this;

/**
 * @var $model Magentostudy_News_Model_News
 */
$model = Mage::getModel('magentostudy_news/news');

// Set up data rows
$dataRows = array(
    array(
```

```
        'title'          => 'Magento Developer Certification Exams Now
Available Worldwide',
        'author'         => 'Beth Gomez',
        'published_at'   => '2011-12-22',
        'content'        => '<p>In October, Magento launched the beta version
of the Magento Certified Developer exam. Dozens of experienced Magento
developers flocked to the X.commerce Innovate 2011 Conference &ndash; and to
several other beta exam locations around the world &ndash; to take the exam.
Many of them earned passing grades, becoming the world&rsquo;s first Magento
Certified Developers.<br /><br />With the successful beta behind us, we are
excited to announce the global launch of our Certification program.
<strong>Now, developers can take the Magento Certified Developer exam at over
5,000 Prometric Testing Centers worldwide.</strong><br /><br />Find out how
to register, prepare for, and take the certification exam on the Magento U
Certification website.<br /><br /><strong>What you should know about Magento
Developer Certification:</strong><br /><br /></p>
<ul>
<li>Magento Developer Certification is the first and only Magento-sponsored
and approved professional certification program.</li>
<li>Magento Developer Certification Exams were developed by a team of experts
from across the Magento ecosystem to accurately test and verify real-world
development skills.</li>
<li>Developers who pass the exam can differentiate themselves in the
marketplace by using the Magento Developer Certification badge on their
resume/CV, and &ndash; starting in early 2012 &ndash; by listing themselves
in the Magento Certification directory, a key resource for finding and
verifying Magento Certified Developers. </li>
</ul>
<p><br />We&rsquo;ve made it easy for developers to purchase vouchers,
download study guides, and register to take the exam. Visit the Magento U
Certification website to learn more about Magento Developer Certification and
how it can help your business.</p>',
    ),

    array(
        'title'          => 'Introducing Magento Enterprise Premium!',
        'author'         => 'Pedram Yasharel',
        'published_at'   => '2011-11-23',
        'content'        => '<p>We have just launched the Magento Enterprise
Premium package, the ultimate packaged solution for large-scale eCommerce
implementations. This package has been tailored specifically for large-scale
eCommerce implementations that need the scale, expertise and support
necessary to run a high volume business.<br /><br />Here are the components
we are pleased to offer, as part of this new, premium solution:<br /><br
/></p>
<ul>
<li>Multiple Magento Enterprise licenses - 2 production and 1 development
license</li>
<li>Platinum level SLA Magento Support with live 24x7 phone support</li>
<li>Magento Expert Consulting - architectural advisory and comprehensive code
review dedicated to your business needs</li>
<li>Training course &ldquo;eCommerce with Magento&rdquo;</li>
</ul>
<p><br />With this new package, merchants get the advantage of the best-in-
class features of Magento Enterprise, such as multi-store fronts with a
single Admin interface, persistent shopping cart, RMA, private sales,
marketing and merchandising tools and so much more, all with the added
```

```
support, consulting, training and scalability to meet the needs of your
eCommerce business.<br /><br />We are very excited to offer this and invite
you to learn more about this new, premium offering.</p>'
    ),

    array(
        'title'          => 'Magento Supports Facebook Open Graph 2.0!',
        'author'         => 'Baruch Toledano',
        'published_at'   => '2011-10-18',
        'content'        => '<p>The advantage of Facebook as a marketing tool
for your online store just became a lot more powerful. Magento has introduced
support for the new <strong>&ldquo;Want&rdquo;</strong> and
<strong>&ldquo;Own&rdquo;</strong> Facebook buttons, driving social based
traffic for all Magento online stores. The social buttons can be easily
installed on product and catalog pages and are provided as a Magento <a
title="core" href="http://www.magentocommerce.com/magento-
connect/Magento+Core/extension/8041/social_facebook" target="_blank">core</a>
extension for Magento Community and Enterprise.<br /><br />Customers can now
express themselves better than ever by adding items they &ldquo;Want&rdquo;
on their Facebook pages for everyone to see and buy for them). Wish lists and
Facebook profiles can now be closely linked.<br /> <br />With the
&ldquo;Own&rdquo; button marketing goes organic as consumers showcase items
they are proud to own. Your customers become an extension of your marketing
efforts as they are now able to answer questions, write reviews and provide
recommendations on the products they own.<br /><br />Magento is the first
eCommerce platform to offer integration for the &ldquo;Want&rdquo; and
&ldquo;Own&rdquo; buttons with the Facebook Open Graph 2.0 extension. Give
your products and your business the latest Facebook connection.</p>'
    ),
);

// Generate news items
foreach ($dataRows as $data) {
    $model->setData($data)->setOrigData()->save();
}
```

Events

By specifying an event in our code we mark a place where we want any listener to join the process and perform some manipulations. In Magento we have the concept of the Observer which is an analog for the Listener in other programming languages. When some event fires, the system looks for every observer registered for the event and calls its predefined (in `config.xml`) method.

The Observer pattern is a powerful tool for plugging in customizations in key flow areas without overriding core classes and their methods. You can find more information about this and other patterns in the Magento Knowledge Base (http://www.magentocommerce.com/wiki/5_-_modules_and_development/0_-_module_development_in_magento/customizing_magento_using_event-observer_method)

Extension Best Practices

- Core module rewrites should be minimized.
- In many situations the Observer pattern can be used to completely avoid any rewrites.

To illustrate how to use events, we will use an observer to monitor whether or not individual news posts have been viewed. If a new post has been created within 3 days (by default), our observer will add a block with a message to inform the user that news was recently added.

This is just an illustration of how to use event-observer functionality in Magento. Same feature may be done more easily in module's code without the use of observer at all.

Configuring our event

Before we start implementing our observer model, we need to configure our event in `config.xml`.

`config.xml (events section)`

```
<events>
    <before_news_item_display>
        <observers>
            <magentostudy_news>
                <class>magentostudy_news/observer</class>
                <method>beforeNewsDisplayed</method>
            </magentostudy_news>
        </observers>
    </before_news_item_display>
</events>
```

Here you will find predefined method `beforeNewsDisplayed` in the method node. This method will be called when the `before_news_item_display` event occurs. Any other module may register on this event and perform additional data manipulations.

Dispatching our event

Now that we have configured our event, we need to dispatch it in our code at the appropriate place. In our case we will fire it in the `indexController viewAction` method. Take a look at lines 62-75 in the code snippet above. First, we fire the event and provide a parameter, and then we load and render the layout. Using this sequence allows us to perform additional actions before displaying a news post. If we want to add an event after a post is shown, we need to add the `Mage::dispatchEvent` after rendering the layout.

Magentostudy_News_IndexController::viewAction()

```
Mage::dispatchEvent('before_news_item_display', array('news_item' =>
$model));

$this->loadLayout();
$itemBlock = $this->getLayout()->getBlock('news.item');
if ($itemBlock) {
    $listBlock = $this->getLayout()->getBlock('news.list');
    if ($listBlock) {
        $page = (int)$listBlock->getCurrentPage() ? (int)$listBlock-
>getCurrentPage() : 1;
    } else {
        $page = 1;
    }
    $itemBlock->setPage($page);
}
$this->renderLayout();
```

Implementing our observer logic

Now we have everything we need to work with our event; all that is left is to implement our observer to catch it. The observer is a model, so we will add it here: `Magentostudy/News/Model/Observer.php`

Magentostudy_News_Model_Observer

```
<?php
/**
 * News module observer
 * @author Magento
 */
class Magentostudy_News_Model_Observer
{
    /**
     * Event before show news item on frontend
     * If specified new post was added recently (term is defined in config)
we'll see message about this on frontend.
     *
     * @param Varien_Event_Observer $observer
     */
    public function beforeNewsDisplayed(Varien_Event_Observer $observer)
    {
        $newsItem = $observer->getEvent()->getNewsItem();
        $currentDate = Mage::app()->getLocale()->date();
        $newsCreatedAt = Mage::app()->getLocale()->date(strtotime($newsItem-
>getCreatedAt()));
        $daysDifference = $currentDate->sub($newsCreatedAt)->getTimestamp() /
(60 * 60 * 24);
        if ($daysDifference < Mage::helper('magentostudy_news')-
>getDaysDifference()) {
            Mage::getSingleton('core/session')-
>addSuccess(Mage::helper('magentostudy_news')->__('Recently added'));
        }
```

```
    }
}
```

Development: The Admin Panel

Unlike the frontend which can have many layouts and themes, the Admin interface structure stays constant across all themes.

Magento makes it easy to implement the most-used operations performed in the Admin interface which are displaying lists of items and organizing user forms. Another helpful feature the Admin interface provides is a centralized list of all system configuration options available for editing. You do not need to write much code to organize config options; Magento takes care of them on its own.

As previously mentioned, our extension Admin interface will allow us to create, edit, save and delete news posts. To provide for this functionality, we will create a menu item to access a list of posts and individual news posts respectively. Also we will add configuration fields for enabling news posts on the frontend, setting the number of posts per page, and setting the number of days a post is considered recent (see the section "Development : The Front End - Preparing our helpers").

Similar to our frontend, we need to prepare some Magento components to implement Admin panel functionality. We will need to implement following:

- Specify Admin panel configuration
- Specify routes for our pages
- Prepare output for our pages
- Prepare auxiliary helpers for additional data manipulation

Before we start implementation of our code, consider this note about structure:

Extension Best Practices

- If your extension works both with the frontend and the Admin panel, place controllers and blocks related to Admin functionality into a subfolder called `adminhtml` to clarify block and controller scope.

Preparing Admin panel configuration

Before implementing any Admin-related features, we need to specify user permissions for them.

Extension Best Practices

- ACL restrictions should be implemented in both the `adminhtml.xml` configuration file and the Admin controller.

The `adminhtml.xml` configuration file will define our menu items and their placement as well as implement ACL instructions for them. This file will be located in the module's `etc` folder with the other module-specific configuration files.

adminhtml.xml

```
<?xml version="1.0"?>
<!--
/**
 * Magento Admin config
 *
 * @author Magento
 */
-->
<config>
    <menu>
         <news translate="title" module="magentostudy_news">
            <title>News</title>
            <sort_order>65</sort_order>
            <children>
                <manage translate="title" module="magentostudy_news">
                    <title>Manage News</title>
                    <action>adminhtml/news</action>
                    <sort_order>50</sort_order>
                </manage>
            </children>
         </news>
    </menu>

    <acl>
        <resources>
            <Admin>
                <children>
                    <news translate="title" module="magentostudy_news">
                        <title>News</title>
                        <sort_order>65</sort_order>
                        <children>
                            <manage translate="title">
                                <title>Manage News</title>
                                <sort_order>0</sort_order>
                                <children>
                                    <save translate="title">
                                        <title>Save News</title>
                                        <sort_order>0</sort_order>
                                    </save>
                                    <delete translate="title">
                                        <title>Delete News</title>
                                        <sort_order>10</sort_order>
```

```
                                    </delete>
                                </children>
                            </manage>
                        </children>
                    </news>
                    <system>
                        <children>
                            <config>
                                <children>
                                    <news translate="title"
module="magentostudy_news">
                                        <title>News Management</title>
                                    </news>
                                </children>
                            </config>
                        </children>
                    </system>
                </children>
            </Admin>
        </resources>
    </acl>
</config>
```

As previously mentioned, we would also like to configure some store-related values such as the number of posts per page. For this, we will create a separate file named `system.xml` and store it in the module `etc` folder. After implementing this file, our fields will appear in System > Configuration > General > News. Tab, label, and field sequence can be specified via this `system.xml` file as well.

system.xml

```
<?xml version="1.0"?>
<!--
/**
 * Magento Admin configuration section config
 * @author Magento
 */
-->
<config>
    <sections>
        <news>
            <class>separator-top</class>
            <label>News</label>
            <tab>general</tab>
            <frontend_type>text</frontend_type>
            <sort_order>500</sort_order>
            <show_in_default>1</show_in_default>
            <show_in_website>1</show_in_website>
            <show_in_store>1</show_in_store>

            <groups>
                <view translate="label">
                    <label>News View Settings</label>
                    <frontend_type>text</frontend_type>
```

```
                    <sort_order>10</sort_order>
                    <show_in_default>1</show_in_default>
                    <show_in_website>1</show_in_website>
                    <show_in_store>1</show_in_store>
                    <fields>
                        <enabled translate="label">
                            <label>Enable News On Frontend</label>
                            <frontend_type>select</frontend_type>

<source_model>adminhtml/system_config_source_yesno</source_model>
                            <sort_order>10</sort_order>
                            <show_in_default>1</show_in_default>
                            <show_in_website>1</show_in_website>
                            <show_in_store>1</show_in_store>
                        </enabled>
                        <items_per_page translate="label">
                            <label>News Per Page</label>
                            <comment>Empty value is the same as default
20.</comment>
                            <sort_order>30</sort_order>
                            <depends><enabled>1</enabled></depends>
                            <show_in_default>1</show_in_default>
                            <show_in_website>1</show_in_website>
                            <show_in_store>1</show_in_store>
                        </items_per_page>
                        <days_difference translate="label">
                            <label>Number of days since a post has been
marked as recently added</label>
                            <comment>Empty value is the same as default
3.</comment>
                            <sort_order>50</sort_order>
                            <depends><enabled>1</enabled></depends>
                            <show_in_default>1</show_in_default>
                            <show_in_website>1</show_in_website>
                            <show_in_store>1</show_in_store>
                        </days_difference>
                    </fields>
                </view>
            </groups>
        </news>
    </sections>
</config>
```

Specifying our routes and request flow

Configuring our routes

Before implementing our controller, we need to configure our Admin routes in `config.xml`.

config.xml (specifying routes)

```
<Admin>
    <routers>
        <adminhtml>
            <args>
                <modules>
                    <Magentostudy_News
before="Mage_adminhtml">Magentostudy_News_adminhtml</Magentostudy_News>
                </modules>
            </args>
        </adminhtml>
    </routers>
</Admin>
```

Implementing our controller

According to our plan, we need the following actions for the Admin interface:

- List of news posts
- New news post
- Edit news post
- Save news post
- Delete news post

Also, we need to implement `gridAction` to make our extension similar to other lists in the Admin interface and the `_isAllowed` method to handle ACL instructions.

We'll add our controller `Magentostudy/News/controller/adminhtml/NewsController.php`

Magentostudy_News_adminhtml_NewsController

```
<?php
/**
 * News controller
 *
 * @author Magento
 */
class Magentostudy_News_adminhtml_NewsController extends
Mage_adminhtml_Controller_Action
{
    /**
     * Init actions
     *
     * @return Magentostudy_News_adminhtml_NewsController
     */
    protected function _initAction()
    {
        // load layout, set active menu and breadcrumbs
        $this->loadLayout()
```

```
            ->_setActiveMenu('news/manage')
            ->_addBreadcrumb(
                  Mage::helper('magentostudy_news')->__('News'),
                  Mage::helper('magentostudy_news')->__('News')
              )
            ->_addBreadcrumb(
                  Mage::helper('magentostudy_news')->__('Manage News'),
                  Mage::helper('magentostudy_news')->__('Manage News')
              )
        ;
        return $this;
    }

    /**
     * Index action
     */
    public function indexAction()
    {
        $this->_title($this->__('News'))
             ->_title($this->__('Manage News'));

        $this->_initAction();
        $this->renderLayout();
    }

    /**
     * Create new News item
     */
    public function newAction()
    {
        // the same form is used to create and edit
        $this->_forward('edit');
    }

    /**
     * Edit News item
     */
    public function editAction()
    {
        $this->_title($this->__('News'))
             ->_title($this->__('Manage News'));

        // 1. instance news model
        /* @var $model Magentostudy_News_Model_Item */
        $model = Mage::getModel('magentostudy_news/news');

        // 2. if ID exists, check it and load data
        $newsId = $this->getRequest()->getParam('id');
        if ($newsId) {
            $model->load($newsId);

            if (!$model->getId()) {
                $this->_getSession()->addError(
                    Mage::helper('magentostudy_news')->__('News item does not
exist.')
                );
                return $this->_redirect('*/*/');
```

```
        }
        // prepare title
        $this->_title($model->getTitle());
        $breadCrumb = Mage::helper('magentostudy_news')->__('Edit Item');
    } else {
        $this->_title(Mage::helper('magentostudy_news')->__('New Item'));
        $breadCrumb = Mage::helper('magentostudy_news')->__('New Item');
    }

    // Init breadcrumbs
    $this->_initAction()->_addBreadcrumb($breadCrumb, $breadCrumb);

    // 3. Set entered data if there was an error during save
    $data = Mage::getSingleton('adminhtml/session')->getFormData(true);
    if (!empty($data)) {
        $model->addData($data);
    }

    // 4. Register model to use later in blocks
    Mage::register('news_item', $model);

    // 5. render layout
    $this->renderLayout();
}

/**
 * Save action
 */
public function saveAction()
{
    $redirectPath   = '*/*';
    $redirectParams = array();

    // check if data sent
    $data = $this->getRequest()->getPost();
    if ($data) {
        $data = $this->_filterPostData($data);
        // init model and set data
        /* @var $model Magentostudy_News_Model_Item */
        $model = Mage::getModel('magentostudy_news/news');

        // if news item exists, try to load it
        $newsId = $this->getRequest()->getParam('news_id');
        if ($newsId) {
            $model->load($newsId);
        }
        // save image data and remove from data array
        if (isset($data['image'])) {
            $imageData = $data['image'];
            unset($data['image']);
        } else {
            $imageData = array();
        }
        $model->addData($data);

        try {
            $hasError = false;
```

```
                /* @var $imageHelper Magentostudy_News_Helper_Image */
                $imageHelper = Mage::helper('magentostudy_news/image');
                // remove image

                if (isset($imageData['delete']) && $model->getImage()) {
                    $imageHelper->removeImage($model->getImage());
                    $model->setImage(null);
                }

                // upload new image
                $imageFile = $imageHelper->uploadImage('image');
                if ($imageFile) {
                    if ($model->getImage()) {
                        $imageHelper->removeImage($model->getImage());
                    }
                    $model->setImage($imageFile);
                }
                // save the data
                $model->save();

                // display success message
                $this->_getSession()->addSuccess(
                    Mage::helper('magentostudy_news')->__('The news item has
been saved.')
                );

                // check if 'Save and Continue'
                if ($this->getRequest()->getParam('back')) {
                    $redirectPath   = '*/*/edit';
                    $redirectParams = array('id' => $model->getId());
                }
            } catch (Mage_Core_Exception $e) {
                $hasError = true;
                $this->_getSession()->addError($e->getMessage());
            } catch (Exception $e) {
                $hasError = true;
                $this->_getSession()->addException($e,
                    Mage::helper('magentostudy_news')->__('An error occurred
while saving the news item.')
                );
            }

            if ($hasError) {
                $this->_getSession()->setFormData($data);
                $redirectPath   = '*/*/edit';
                $redirectParams = array('id' => $this->getRequest()-
>getParam('id'));
            }
        }

        $this->_redirect($redirectPath, $redirectParams);
    }

    /**
     * Delete action
     */
    public function deleteAction()
```

```
    {
        // check if we know what should be deleted
        $itemId = $this->getRequest()->getParam('id');
        if ($itemId) {
            try {
                // init model and delete
                /** @var $model Magentostudy_News_Model_Item */
                $model = Mage::getModel('magentostudy_news/news');
                $model->load($itemId);
                if (!$model->getId()) {
                    Mage::throwException(Mage::helper('magentostudy_news')-
>__('Unable to find a news item.'));
                }
                $model->delete();

                // display success message
                $this->_getSession()->addSuccess(
                    Mage::helper('magentostudy_news')->__('The news item has
been deleted.')
                );
            } catch (Mage_Core_Exception $e) {
                $this->_getSession()->addError($e->getMessage());
            } catch (Exception $e) {
                $this->_getSession()->addException($e,
                    Mage::helper('magentostudy_news')->__('An error occurred
while deleting the news item.')
                );
            }
        }

        // go to grid
        $this->_redirect('*/*/');
    }

    /**
     * Check the permission to run it
     *
     * @return boolean
     */
    protected function _isAllowed()
    {
        switch ($this->getRequest()->getActionName()) {
            case 'new':
            case 'save':
                return Mage::getSingleton('Admin/session')-
>isAllowed('news/manage/save');
                break;
            case 'delete':
                return Mage::getSingleton('Admin/session')-
>isAllowed('news/manage/delete');
                break;
            default:
                return Mage::getSingleton('Admin/session')-
>isAllowed('news/manage');
                break;
        }
    }
```

```
    /**
     * Filtering posted data. Converting localized data if needed
     *
     * @param array
     * @return array
     */
    protected function _filterPostData($data)
    {
        $data = $this->_filterDates($data, array('time_published'));
        return $data;
    }

    /**
     * Grid ajax action
     */
    public function gridAction()
    {
        $this->loadLayout();
        $this->renderLayout();
    }

    /**
     * Flush News Posts Images Cache action
     */
    public function flushAction()
    {
        if (Mage::helper('magentostudy_news/image')->flushImagesCache()) {
            $this->_getSession()->addSuccess('Cache successfully flushed');
        } else {
            $this->_getSession()->addError('There was error during flushing
cache');
        }
        $this->_forward('index');
    }
}
```

Preparing our output

As mentioned above, if we do not need a particular piece of output we should not add it. In our case, all we need for the Admin are layout and blocks

Configuring our output

Similarly to the frontend, we need to configure our output before we implement it. We specified the blocks for this when we configured our frontend blocks earlier, so we will just configure our `adminhtml` layout in `config.xml`.

`config.xml (specifying adminhtml layout)`

```
<adminhtml>
    <layout>
        <updates>
```

```
            <magentostudy_news>
                <file>magentostudy_news.xml</file>
            </magentostudy_news>
        </updates>
    </layout>
</adminhtml>
```

Implementing our layout

As shown above, our controller's actions load and render a layout. This layout doesn't exist yet, so next we'll create it.

Magentostudy_news.xml

```
<?xml version="1.0"?>
<!--
/**
 * Magento backend layout
 *
 * @author Magento
 */
-->
<layout>
    <adminhtml_news_index>
        <reference name="content">
            <block type="magentostudy_news/adminhtml_news" name="news" />
        </reference>
    </adminhtml_news_index>

    <adminhtml_news_grid>
        <block type="magentostudy_news/adminhtml_news_grid" name="root"/>
    </adminhtml_news_grid>

    <adminhtml_news_new>
        <update handle="adminhtml_news_edit" />
    </adminhtml_news_new>

    <adminhtml_news_edit>
        <update handle="editor"/>
        <reference name="content">
            <block type="magentostudy_news/adminhtml_news_edit"
name="news_edit" />
        </reference>
        <reference name="left">
            <block type="magentostudy_news/adminhtml_news_edit_tabs"
name="news_edit_tabs">
                <block type="magentostudy_news/adminhtml_news_edit_tab_main"
name="news_edit_tab_main" />
                <block
type="magentostudy_news/adminhtml_news_edit_tab_content"
name="news_edit_tab_content" />
                <block type="magentostudy_news/adminhtml_news_edit_tab_image"
name="news_edit_tab_image" />
```

```
                <action
method="addTab"><name>main_section</name><block>news_edit_tab_main</block></a
ction>
                <action
method="addTab"><name>content_section</name><block>news_edit_tab_content</blo
ck></action>
                <action
method="addTab"><name>image_section</name><block>news_edit_tab_image</block><
/action>
            </block>
        </reference>
    </adminhtml_news_edit>
</layout>
```

Implementing our blocks

As mentioned above, the Admin panel has a similar structure for all of its components. The blocks structure is also similar; they all use a grid that is connected by edit forms, each of which has different tabs.

To illustrate our Admin panel blocks structure, observe the following schema (Figure 6):

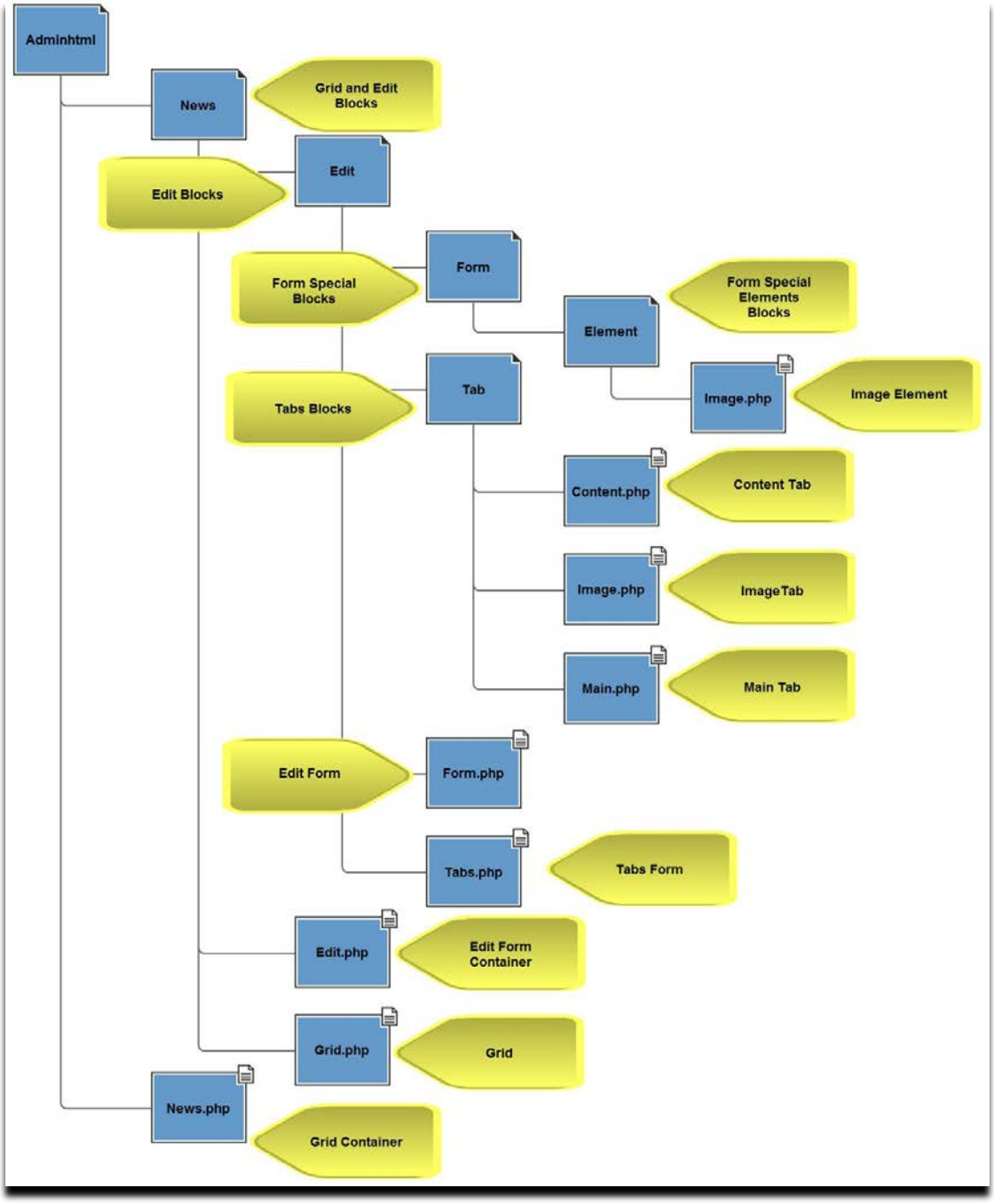

Figure 6: Our Admin panel block structure

We will use the following blocks:

- `adminhtml_news` - grid container
- `adminhtml_news_grid` - grid
- `adminhtml_news_edit` - edit form container (the same forms are used for creating a new post and editing an existing post)

- `adminhtml_news_edit_form` - edit form
- `adminhtml_news_edit_tabs` - edit form tabs
- `adminhtml_news_edit_form_element_image` - specific form element to work with images
- `adminhtml_news_edit_tab_content` - edit form content tab
- `adminhtml_news_edit_tab_image` - edit form image tab
- `adminhtml_news_edit_tab_main` - edit form main tab

Here's our Admin panel block implementation:

Magentostudy_News_Block_adminhtml_News

```
<?php
/**
 * News List Admin grid container
 *
 * @author Magento
 */
class Magentostudy_News_Block_adminhtml_News extends
Mage_adminhtml_Block_Widget_Grid_Container
{
    /**
     * Block constructor
     */
    public function __construct()
    {
        $this->_blockGroup = 'magentostudy_news';
        $this->_controller = 'adminhtml_news';
        $this->_headerText = Mage::helper('magentostudy_news')->__('Manage
News');

        parent::__construct();

        if (Mage::helper('magentostudy_news/Admin')->isActionAllowed('save'))
{
            $this->_updateButton('add', 'label',
Mage::helper('magentostudy_news')->__('Add New News'));
        } else {
            $this->_removeButton('add');
        }
        $this->addButton(
            'news_flush_images_cache',
            array(
                'label'      => Mage::helper('magentostudy_news')->__('Flush
Images Cache'),
                'onclick'    => 'setLocation(\'' . $this->getUrl('*/*/flush')
. '\')',
            )
        );

    }
}
```

Magentostudy_News_Block_adminhtml_News_Edit

```
<?php
/**
 * News List Admin edit form container
 *
 * @author Magento
 */
class Magentostudy_News_Block_adminhtml_News_Edit extends
Mage_adminhtml_Block_Widget_Form_Container
{
    /**
     * Initialize edit form container
     *
     */
    public function __construct()
    {
        $this->_objectId   = 'id';
        $this->_blockGroup = 'magentostudy_news';
        $this->_controller = 'adminhtml_news';

        parent::__construct();

        if (Mage::helper('magentostudy_news/Admin')->isActionAllowed('save'))
{
            $this->_updateButton('save', 'label',
Mage::helper('magentostudy_news')->__('Save News Item'));
            $this->_addButton('saveandcontinue', array(
                'label'   => Mage::helper('adminhtml')->__('Save and Continue
Edit'),
                'onclick' => 'saveAndContinueEdit()',
                'class'   => 'save',
            ), -100);
        } else {
            $this->_removeButton('save');
        }

        if (Mage::helper('magentostudy_news/Admin')-
>isActionAllowed('delete')) {
            $this->_updateButton('delete', 'label',
Mage::helper('magentostudy_news')->__('Delete News Item'));
        } else {
            $this->_removeButton('delete');
        }

        $this->_formScripts[] = "
            function toggleEditor() {
                if (tinyMCE.getInstanceById('page_content') == null) {
                    tinyMCE.execCommand('mceAddControl', false,
'page_content');
                } else {
                    tinyMCE.execCommand('mceRemoveControl', false,
'page_content');
                }
            }
```

```
            function saveAndContinueEdit(){
                editForm.submit($('edit_form').action+'back/edit/');
            }
        ";
    }

    /**
     * Retrieve text for header element depending on loaded page
     *
     * @return string
     */
    public function getHeaderText()
    {
        $model = Mage::helper('magentostudy_news')->getNewsItemInstance();
        if ($model->getId()) {
            return Mage::helper('magentostudy_news')->__("Edit News Item
'%s'",
                 $this->escapeHtml($model->getTitle()));
        } else {
            return Mage::helper('magentostudy_news')->__('New News Item');
        }
    }
}
```

Magentostudy_News_Block_adminhtml_News_Grid

```
<?php
/**
 * News List Admin grid
 *
 * @author Magento
 */
class Magentostudy_News_Block_adminhtml_News_Grid extends
Mage_adminhtml_Block_Widget_Grid
{
    /**
     * Init Grid default properties
     *
     */
    public function __construct()
    {
        parent::__construct();
        $this->setId('news_list_grid');
        $this->setDefaultSort('created_at');
        $this->setDefaultDir('DESC');
        $this->setSaveParametersInSession(true);
        $this->setUseAjax(true);
    }

    /**
     * Prepare collection for Grid
     *
     * @return Magentostudy_News_Block_adminhtml_Grid
     */
    protected function _prepareCollection()
```

```
    {
        $collection = Mage::getModel('magentostudy_news/news')-
>getResourceCollection();

        $this->setCollection($collection);
        return parent::_prepareCollection();
    }

    /**
     * Prepare Grid columns
     *
     * @return Mage_adminhtml_Block_Catalog_Search_Grid
     */
    protected function _prepareColumns()
    {
        $this->addColumn('news_id', array(
            'header'    => Mage::helper('magentostudy_news')->__('ID'),
            'width'     => '50px',
            'index'     => 'news_id',
        ));

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

        $this->addColumn('author', array(
            'header'    => Mage::helper('magentostudy_news')->__('Author'),
            'index'     => 'author',
        ));

        $this->addColumn('published_at', array(
            'header'   => Mage::helper('magentostudy_news')->__('Published
On'),
            'sortable' => true,
            'width'    => '170px',
            'index'    => 'published_at',
            'type'     => 'date',
        ));

        $this->addColumn('created_at', array(
            'header'   => Mage::helper('magentostudy_news')->__('Created'),
            'sortable' => true,
            'width'    => '170px',
            'index'    => 'created_at',
            'type'     => 'datetime',
        ));

        $this->addColumn('action',
            array(
                'header'    => Mage::helper('magentostudy_news')-
>__('Action'),
                'width'     => '100px',
                'type'      => 'action',
                'getter'    => 'getId',
                'actions'   => array(array(
```



```
                    'caption' => Mage::helper('magentostudy_news')-
>__('Edit'),
                    'url'     => array('base' => '*/*/edit'),
                    'field'   => 'id'
                )),
                'filter'    => false,
                'sortable'  => false,
                'index'     => 'news',
        ));

        return parent::_prepareColumns();
    }

    /**
     * Return row URL for js event handlers
     *
     * @return string
     */
    public function getRowUrl($row)
    {
        return $this->getUrl('*/*/edit', array('id' => $row->getId()));
    }

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

Magentostudy_News_Block_adminhtml_News_Edit_Form

```
<?php
/**
 * News List Admin edit form block
 *
 * @author Magento
 */
class Magentostudy_News_Block_adminhtml_News_Edit_Form extends
Mage_adminhtml_Block_Widget_Form
{
    /**
     * Prepare form action
     *
     * @return Magentostudy_News_Block_adminhtml_News_Edit_Form
     */
    protected function _prepareForm()
    {
        $form = new Varien_Data_Form(array(
            'id'      => 'edit_form',
            'action'  => $this->getUrl('*/*/save'),
            'method'  => 'post',
```

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

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

Magentostudy_News_Block_adminhtml_News_Edit_Tabs

```
<?php
/**
 * News List Admin edit form tabs block
 *
 * @author Magento
 */
class Magentostudy_News_Block_adminhtml_News_Edit_Tabs extends
Mage_adminhtml_Block_Widget_Tabs
{
    /**
     * Initialize tabs and define tabs block settings
     *
     */
    public function __construct()
    {
        parent::__construct();
        $this->setId('page_tabs');
        $this->setDestElementId('edit_form');
        $this->setTitle(Mage::helper('magentostudy_news')->__('News Item
Info'));
    }
}
```

Magentostudy_News_Block_adminhtml_News_Edit_Form_Element_Image

```
<?php
/**
 * Custom image form element that generates correct thumbnail image URL
 *
 * @author Magento
 */
class Magentostudy_News_Block_adminhtml_News_Edit_Form_Element_Image extends
Varien_Data_Form_Element_Image
{
    /**
     * Get image preview url
     *
     * @return string
     */
    protected function _getUrl()
    {
        $url = false;
```

```
        if ($this->getValue()) {
            $url = Mage::helper('magentostudy_news/image')->getBaseUrl() .
'/' . $this->getValue();
        }
        return $url;
    }
}
```

Magentostudy_News_Block_adminhtml_News_Edit_Tab_Content

```
<?php
/**
 * News List Admin edit form content tab
 *
 * @author Magento
 */
class Magentostudy_News_Block_adminhtml_News_Edit_Tab_Content
    extends Mage_adminhtml_Block_Widget_Form
    implements Mage_adminhtml_Block_Widget_Tab_Interface
{
    /**
     * Load WYSIWYG on demand and prepare layout
     *
     * @return Magentostudy_News_Block_adminhtml_News_Edit_Tab_Content
     */
    protected function _prepareLayout()
    {
        parent::_prepareLayout();
        if (Mage::getSingleton('cms/wysiwyg_config')->isEnabled()) {
            $this->getLayout()->getBlock('head')->setCanLoadTinyMce(true);
        }
        return $this;
    }

    /**
     * Prepares tab form
     *
     * @return Mage_adminhtml_Block_Widget_Form
     */
    protected function _prepareForm()
    {
        $model = Mage::helper('magentostudy_news')->getNewsItemInstance();

        /**
         * Checking if user have permissions to save information
         */
        if (Mage::helper('magentostudy_news/Admin')->isActionAllowed('save'))
{
            $isElementDisabled = false;
        } else {
            $isElementDisabled = true;
        }

        $form = new Varien_Data_Form();

        $form->setHtmlIdPrefix('news_content_');
```

```
        $fieldset = $form->addFieldset('content_fieldset', array(
            'legend' => Mage::helper('magentostudy_news')->__('Content'),
            'class'  => 'fieldset-wide'
        ));

        $wysiwygConfig = Mage::getSingleton('cms/wysiwyg_config')-
>getConfig(array(
            'tab_id' => $this->getTabId()
        ));

        $contentField = $fieldset->addField('content', 'editor', array(
            'name'     => 'content',
            'style'    => 'height:36em;',
            'required' => true,
            'disabled' => $isElementDisabled,
            'config'   => $wysiwygConfig
        ));

        // Setting custom renderer for content field to remove label column
        $renderer = $this->getLayout()-
>createBlock('adminhtml/widget_form_renderer_fieldset_element')
            ->setTemplate('cms/page/edit/form/renderer/content.phtml');
        $contentField->setRenderer($renderer);

        $form->setValues($model->getData());
        $this->setForm($form);

        Mage::dispatchEvent('adminhtml_news_edit_tab_content_prepare_form',
array('form' => $form));

        return parent::_prepareForm();
    }

    /**
     * Prepare label for tab
     *
     * @return string
     */
    public function getTabLabel()
    {
        return Mage::helper('magentostudy_news')->__('Content');
    }

    /**
     * Prepare title for tab
     *
     * @return string
     */
    public function getTabTitle()
    {
        return Mage::helper('magentostudy_news')->__('Content');
    }

    /**
     * Returns status flag about this tab can be shown or not
     *
```

```
     * @return true
     */
    public function canShowTab()
    {
        return true;
    }

    /**
     * Returns status flag about this tab hidden or not
     *
     * @return true
     */
    public function isHidden()
    {
        return false;
    }
}
```

Magentostudy_News_Block_adminhtml_News_Edit_Tab_Image

```
<?php
/**
 * News List Admin edit form image tab
 *
 * @author Magento
 */
class Magentostudy_News_Block_adminhtml_News_Edit_Tab_Image
    extends Mage_adminhtml_Block_Widget_Form
    implements Mage_adminhtml_Block_Widget_Tab_Interface
{
    /**
     * Prepare form elements
     *
     * @return Mage_adminhtml_Block_Widget_Form
     */
    protected function _prepareForm()
    {
        /**
         * Checking if user have permissions to save information
         */
        if (Mage::helper('magentostudy_news/Admin')->isActionAllowed('save'))
{
            $isElementDisabled = false;
        } else {
            $isElementDisabled = true;
        }

        $form = new Varien_Data_Form();

        $form->setHtmlIdPrefix('news_image_');

        $model = Mage::helper('magentostudy_news')->getNewsItemInstance();


        $fieldset = $form->addFieldset('image_fieldset', array(
```

```
            'legend'    => Mage::helper('magentostudy_news')->__('Image
Thumbnail'), 'class' => 'fieldset-wide'
        ));

        $this->_addElementTypes($fieldset);

        $fieldset->addField('image', 'image', array(
            'name'      => 'image',
            'label'     => Mage::helper('magentostudy_news')->__('Image'),
            'title'     => Mage::helper('magentostudy_news')->__('Image'),
            'required'  => false,
            'disabled'  => $isElementDisabled
        ));

        Mage::dispatchEvent('adminhtml_news_edit_tab_image_prepare_form',
array('form' => $form));

        $form->setValues($model->getData());
        $this->setForm($form);

        return parent::_prepareForm();
    }

    /**
     * Prepare label for tab
     *
     * @return string
     */
    public function getTabLabel()
    {
        return Mage::helper('magentostudy_news')->__('Image Thumbnail');
    }

    /**
     * Prepare title for tab
     *
     * @return string
     */
    public function getTabTitle()
    {
        return Mage::helper('magentostudy_news')->__('Image Thumbnail');
    }

    /**
     * Returns status flag about this tab can be showen or not
     *
     * @return true
     */
    public function canShowTab()
    {
        return true;
    }

    /**
     * Returns status flag about this tab hidden or not
     *
     * @return true
```

```
     */
    public function isHidden()
    {
        return false;
    }

    /**
     * Retrieve predefined additional element types
     *
     * @return array
     */
     protected function _getAdditionalElementTypes()
     {
         return array(
            'image' => Mage::getConfig()-
>getBlockClassName('magentostudy_news/adminhtml_news_edit_form_element_image'
)
        );
     }
}
```

Magentostudy_News_Block_adminhtml_News_Edit_Tab_Main

```
<?php
/**
 * News List Admin edit form main tab
 *
 * @author Magento
 */
class Magentostudy_News_Block_adminhtml_News_Edit_Tab_Main
    extends Mage_adminhtml_Block_Widget_Form
    implements Mage_adminhtml_Block_Widget_Tab_Interface
{
    /**
     * Prepare form elements for tab
     *
     * @return Mage_adminhtml_Block_Widget_Form
     */
    protected function _prepareForm()
    {
        $model = Mage::helper('magentostudy_news')->getNewsItemInstance();

        /**
         * Checking if user have permissions to save information
         */
        if (Mage::helper('magentostudy_news/Admin')->isActionAllowed('save'))
{
            $isElementDisabled = false;
        } else {
            $isElementDisabled = true;
        }

        $form = new Varien_Data_Form();

        $form->setHtmlIdPrefix('news_main_');
```

```
        $fieldset = $form->addFieldset('base_fieldset', array(
            'legend' => Mage::helper('magentostudy_news')->__('News Item
Info')
        ));

        if ($model->getId()) {
            $fieldset->addField('news_id', 'hidden', array(
                'name' => 'news_id',
            ));
        }

        $fieldset->addField('title', 'text', array(
            'name'     => 'title',
            'label'    => Mage::helper('magentostudy_news')->__('News
Title'),
            'title'    => Mage::helper('magentostudy_news')->__('News
Title'),
            'required' => true,
            'disabled' => $isElementDisabled
        ));

        $fieldset->addField('author', 'text', array(
            'name'     => 'author',
            'label'    => Mage::helper('magentostudy_news')->__('Author'),
            'title'    => Mage::helper('magentostudy_news')->__('Author'),
            'required' => true,
            'disabled' => $isElementDisabled
        ));

        $fieldset->addField('published_at', 'date', array(
            'name'     => 'published_at',
            'format'   => Mage::app()->getLocale()-
>getDateFormat(Mage_Core_Model_Locale::FORMAT_TYPE_SHORT),
            'image'    => $this->getSkinUrl('images/grid-cal.gif'),
            'label'    => Mage::helper('magentostudy_news')->__('Publishing
Date'),
            'title'    => Mage::helper('magentostudy_news')->__('Publishing
Date'),
            'required' => true
        ));

        Mage::dispatchEvent('adminhtml_news_edit_tab_main_prepare_form',
array('form' => $form));

        $form->setValues($model->getData());
        $this->setForm($form);

        return parent::_prepareForm();
    }

    /**
     * Prepare label for tab
     *
     * @return string
     */
    public function getTabLabel()
    {
```

```
        return Mage::helper('magentostudy_news')->__('News Info');
    }

    /**
     * Prepare title for tab
     *
     * @return string
     */
    public function getTabTitle()
    {
        return Mage::helper('magentostudy_news')->__('News Info');
    }

    /**
     * Returns status flag about this tab can be shown or not
     *
     * @return true
     */
    public function canShowTab()
    {
        return true;
    }

    /**
     * Returns status flag about this tab hidden or not
     *
     * @return true
     */
    public function isHidden()
    {
        return false;
    }
}
```

Preparing our helpers

As mentioned in the frontend helpers section, helpers are auxiliary classes that implement commonly used business logic. In the case of our Admin panel, we'll use helpers to check permissions.

We have one helper, `Admin.php`, with the following code:

`Magentostudy_News_Helper_Admin`

```
<?php
/**
 * News Admin helper
 *
 * @author Magento
 */
class Magentostudy_News_Helper_Admin extends Mage_Core_Helper_Abstract
{
    /**
     * Check permission for passed action
```

```
     *
     * @param string $action
     * @return bool
     */
    public function isActionAllowed($action)
    {
        return Mage::getSingleton('Admin/session')->isAllowed('news/manage/'
. $action);
    }
}
```

Download the Extension

You can download the extension we created in the process of documenting its steps from the following location:

Most recent archived version: Magentostudy_News.zip [http://info.magento.com/rs/magentocommerce/images/Magentostudy_News.zip]

Version to install via Magento Connect: Magentostudy_News-1.0.0.0.1.tgz [http://info.magento.com/rs/magentocommerce/images/Magentostudy_News-1.0.0.0.1.tgz]

Appendix A: Building a Trusted Extension

Overview

Magento is a feature-rich eCommerce platform built on open-source technology that provides online merchants with unprecedented flexibility and control over the look, content, and functionality of their eCommerce store. To allow for such flexibility, extension developers must follow the following guidelines. Those extensions that do not follow these guidelines might not be accepted for certification.

Extension Structure and Configuration

1. Code for extensions that are intended for community use should be stored in the `community` code pool.
2. Frontend templates and layouts should be stored in the `frontend/base/default` package.
3. Admin panel templates and layouts should be stored in the: `adminhtml/default/default` package.
4. There are two locations that can be used to store JavaScript files -- in the application `js` folder or an appropriate `skin` folder. If you use the application `js` folder, JavaScript will need to be stored in a second nested folder that is named according to the module or library to which the js applies. This is independent of scope (frontend or Admin). If you use the `skin` folder, frontend JavaScript should be stored in `skin/frontend/base/default/js/<namespace>_<name>`, Admin panel JavaScript should be stored in `skin/adminhtml/default/default/js/<namespace>_<name>`.
5. Extension-specific CSS files should be stored in `css` folders within an appropriate `skin` folder depending on whether they apply to the frontend or the Admin panel.
    - Frontend CSS: `skin/frontend/base/default/css`.
    - Admin panel CSS: `skin/adminhtml/default/default/css`.
6. Extension-specific JavaScript and CSS files should not overwrite any native files.
7. Images and other media files should be stored in the `media` folder. If you supply media files within the extension package, they will need to be stored in an appropriate `skin` folder, otherwise they will not be found by the code.
8. Always be sure to check your module's dependencies, because a missing or incorrectly defined dependency could cause your extension to produce fatal errors. Only necessary dependencies should be specified, as they influence the

loading sequence of the application's core modules. Unnecessary dependences should not be included.
9. The configuration file and the Admin controller should implement ACL restrictions.

Using the Database, Resources, and Data

1. Do not create references from core database tables to extension ones because this can break native Magento functionality.
2. Data should be escaped before inserting it into the database. This is extremely important for security because it helps to avoid SQL-injections.
3. The extension should be responsible for validating incoming data such as user entries, API call responses, etc.
4. Be sure to use the correct connection for queries. If you want only select records – use read connection, otherwise use write connection. Pay attention that all queries like "SELECT … FOR UPDATE" also need write permission, so write connection should be used. Pay attention to queries combined into transactions – if only one query needs write permissions, all queries in the transaction should use write connection.
5. It is forbidden to alter database structure on the fly. All changes should be applied via install scripts. If the extension needs some temporary database tables, they should be preliminarily created via install scripts and cleaned in appropriate situations.

Working with Output

1. Core templates files cannot be physically overridden in the extension package. Use layout functionality to extend frontend views.
2. All phrases must be enclosed with translation tags and methods so that they will be translated according to the store locale.
3. All data shown in templates should be preliminarily escaped with the appropriate methods (for example `escapeHtml`, `stripTags`) to avoid possible XSS injections.

Magento Components and Coding Style

1. Do not use PHP global variables like `$_SERVER`, `$_GET`, `$_POST`. There are special Magento methods to interact with this data that should be used instead.
2. An extension should have its own helper for translations, to allow different phrase translations in different modules and to make the module independent from others.
3. Do not work directly with PHP `$_GLOBAL` variable, Magento provides alternative solution to avoid usage of `$_GLOBAL - Mage::register(), Mage::registry(), Mage::unregister()`.

Appendix B: Magento Best Practices

Overview

Magento is a feature-rich eCommerce platform built on open-source technology that provides online merchants with unprecedented flexibility and control over the look, content and functionality of their eCommerce store. To ensure that extensions are compatible, flexible, clear, secure, and performance optimized, developers should follow the best practices specified below.

Extension Structure and Configuration

1. In order to improve readability and make the relationship between templates and the module code more clear, templates should be stored in separate folders that are named with the module namespace and name.
2. For similar reasons, layouts should be named with the module namespace and name. Note, however, that native layouts don't include a namespace in their names.
3. If your extension works both with frontend and Admin panel, place controllers and blocks related to Admin functionality into a sub folder `adminhtml` to clarify their scope.
4. External libraries needed by an extension should be stored in the `lib` folder and should be open source.

Using the Database, Resources, and Data

1. In order to improve readability and make the relationship between database tables and code more clearly, database tables should be named by module namespace and name.
2. Use database indexes when creating tables because this can significantly improve database (and overall) performance. More than one index can be created for a table.
3. Data and structure upgrades should be processed via Adapter methods; raw SQL is not recommended. Usage of adapter methods will make an extension independent of concrete database realization.
4. If the extension stores some data such as statistics, the best approach is to create the necessary logic using database storage instead of the file system.
5. Separate data upgrades from schema upgrades. Pay strict attention to this because different upgrades were added in Magento Enterprise Edition (EE) 1.11 and Magento Community Edition (CE) 1.6.
6. Database query manipulations should be placed in resource models only. Resource models act as an abstraction layer between domain models and the database so they are the best place for SQL.

7. Use Zend style SQL queries because they make SQL queries more readable and provide an object-oriented approach, which allows modifying them in other modules.

Working with Output

1. It is not recommended to remove native blocks in custom layouts. The best way is to add new blocks.
2. Frontend Blocks should be cached, wherever possible, to optimize frontend performance. All blocks except special dynamic content should be cached using native Magento cache functionality. Use logical caching when possible.
3. Templates should not contain business logic. Move all business logic to models, blocks, and helpers.

Magento Components and Coding Style

1. Override controllers only if you do not have another more flexible solution.
2. If you plan to just add new action(s) without using methods or variables from the parent, use an abstract class for the parent. This will help keep your extension compatible with others.
3. According to the Zend Framework MVC best practices, the controller component of an application "must be as skinny as possible." It is strongly recommended to encapsulate all business logic into models and helpers; avoid implementing business logic in a controller action.
4. Do not hardcode pre-defined values directly into methods. Use class constants, protected class variables, or store configuration to allow for changing values without code changes.
5. Core module rewrites should be minimized. In many situations the Observer pattern can be used to completely avoid any rewrites.
6. Do not use the version of Magento in extension logic to change flow of the logic. If it is needed universal extension for several versions of Magento, specific module version should be checked. This has been explained in more detail below:

   Consider that the extension is being developed to support different versions of Magento, for example CE 1.5 and CE 1.6. According to Magento versions code changes, the extension should use different algorithms, and major Magento version (1.5.0.0 or 1.6.0.1) shouldn't be checked. Each module has its version that is identical for Magento CE, EE and Go (after merge codebase) and should be used in such cases.

   For example, the extension works with Sales module. The database structure was changed in EE 1.6, but extension wants to work properly on new and old versions, so it has two algorithms that depend on Magento version. Currently, extension developers are checking this like in the code snippet below:

```
if (Mage::getVersion() >= 1.6) {
   ... algorithm 1
} else {
   ... algorithm 2
}
```

But this is incorrect because EE 1.6 and CE 1.6 have the identical value as the result of the call function Mage::getVersion. The sales module version should be checked to decide which algorithm should be used.

Appendix C: Further References

- Designer's Guide to Magento
  - http://info.magento.com/rs/magentocommerce/images/MagentoDesignGuide.pdf

- How to Package Magento Connect Extensions
  - http://info.magento.com/rs/magentocommerce/images/packagingmagentoconnectextensions6%200.pdf

- Alan Storm: Magento Articles for Professional Developers
  - http://alanstorm.com/category/magento#magento_for_php_developers

- Inchoo. Magento Articles
  - http://inchoo.net/category/ecommerce/magento/