

### ⚙️ Design Patterns In Magento

#### 🔗 Model View Controller Pattern (MVC)

Design pattern where business, presentation and coupling logic are separated

#### 🔗 Front Controller Pattern

Makes sure that there is one and only one point of entry. All requests are investigated, routed to the designated controller and then processed accordingly to the specification

#### 🔗 Factory Pattern

Responsible of factorizing (instantiating) classes

#### 🔗 Singleton Pattern

Checks the whether this class has already been instantiated before, this results in a shared instance.

#### 🔗 Registry Pattern

Internal registry: a global scoped container for storing data

#### 🔗 Prototype Pattern

It defines that instances of classes can retrieve a specific other class instance depending on its parent class (the prototype)

#### 🔗 Object Pool Pattern

A box with objects so that they do not have to be allocated and destroyed over and over again

#### 🔗 Iterator Pattern

The iterator pattern defines that there is a shared way to iterate over a container with objects

#### 🔗 Lazy Loading Pattern

Lazy loading ensures that loading data is delayed until the point when it is actually needed

#### 🔗 Service Locator Pattern

The service locator pattern abstracts away the retrieval of a certain service. This allows for changing the service without breaking anything

#### 🔗 Module Pattern

An implementation of the module pattern would make sure that each element can be removed or swapped

#### 🔗 Observer Pattern

By defining observers (or listeners), extra code can be hooked which will be called upon as the observed event fires

#### 🔗 Proxy Pattern

### ⚙️ Design Patterns In Magento (cont)

🔗 Provide a surrogate or placeholder for another object to control access to it.

🔗 Use an extra level of indirection to support distributed, controlled, or intelligent access.

🔗 Add a wrapper and delegation to protect the real component from undue complexity.

#### 📄 MODULE.XML File

A component declares itself (that is, defines its name and existence) in

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.magento.com/etc/module.xsd">
<module name="Vendor_Component_Name" setup_version="1.0">
<sequence><module name="Magento_Sales"/></sequence>
</config>
```

#### 📄 CONFIG.XML File

Used to configure module. The responsibilities of the **config.xml** configuration

##### Example

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.magento.com/etc/module.xsd"><default>
<customsetting><general><enable>1</enable><title>Module Title</title>
</customsetting></default></config>
```

#### 📄 DI.XML File

What case we use di.xml ?

##### Example for Preference

```
<preference for="Magento\Customer\ViewModel" to="Vendor\Custom\ViewModel" />
```

Above code, When someone asks to instantiate a `Magento\Customer\ViewModel`

##### Example for Arguments

```
<type name="Magento\Customer\Model\Cart" argument="customer" value="customer" />
```



By **Abdellatif EL MIZEB**  
(latmiz)  
[cheatography.com/latmiz/](https://cheatography.com/latmiz/)

Published 21st February, 2017.  
Last updated 21st February, 2017.  
Page 1 of 4.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish  
Yours!  
<https://apollopad.com>

### DI.XML File (cont)

In the above code, We are sending object as an argument, we are saying system proxy class with the data as group flag

#### Example for Plugin

```
<type name="Magento_Customer_Model_Resource_Modul_Visitor"><plugin name="Customer_Log" /></type>
```

In the above code, public function clean(\$object) in visitor class is called after public function afterClean(\$subject, \$resourceModel) written in A menu.xml file is a collection of <add/> nodes. Each of these nodes ac

#### Example for Virtual Types

```
<virtualType name="ourVirtualTypeName" type="Pulsestorm_Tutorial_VirtualType">
```

Creating a virtual type is sort of like creating a sub-class for an existing class. With virtual types, the only behavior you can change in your virtual st

### EVENTS.XML File

Custom events can be dispatched by simply passing in a unique event name to the event manager when you call the dispatch function. The unique event name is referenced in your module's **events.xml** file where you specify which observers will react to that event.

The observer xml element has the following properties:

<b>name (required)</b>	The name of the observer for the event definition
<b>instance (required)</b>	The fully qualified class name of the observer
<b>disabled</b>	Determines whether this observer is active
<b>shared</b>	Determines the lifecycle of the class. Default is false

#### Example

```
<event name="my_module_event_before">
<observer name="myObserverName" instance="MyObserver" />
</event>
```

### ACL.XML File

file is where we define our module access control list resources. Access control list resources are visible under the Magento admin System | Permi

#### Example

```
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="etc/webapi.xsd">
<acl><resources><resource id="Magento_Backend::admin">
<resource id="Magento_Backend::jobs" title="Jobs" sortOrder="100">
<resource id="Magento_Backend::jobs::delete_department" title="Delete Department" sortOrder="20" />
</resource></resources></acl>
</config>
```

### SYSTEM.XML File

The system.xml file is used to define configuration fields in Magento 2 System Configuration. You will need this if your module has some settings which the admin needs to set

### MENU.XML File

A menu.xml file is a collection of <add/> nodes. Each of these nodes ac

#### Example

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="etc/menu.xml">
<menu><add id="ExampleMenuTutorial::my_menu" title="Example Menu" sortOrder="10" resource="Magento_Backend::content" /></menu>
</config>
```

<b>id</b>	Attribute defines a unique identifier for this node.
<b>title</b>	Attribute controls the text an end user sees for the menu item.
<b>module</b>	Attribute should match the current module
<b>sortOrder</b>	Attribute controls how this Menu Item is sorted with other menu items.
<b>resource</b>	attribute defines the ACL rule a user must have in order to view the menu item.

### WIDGET.XML File

The file that is responsible for widgets in Magento is placed in [module\_dir]/etc/widget.xml

### WEBAPI.XML File

The **webapi.xml** is used to configure access rights and API Interface with

#### Example

```
<?xml version="1.0"?>
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="etc/webapi.xsd">
<route url="/v1/hello/name/:name" method="GET">
<service class="HelloApi\HelloInterface">
<resource id="HelloApi::hello" />
</service>
</route></routes>
```

Resource tag defines what resources user needs to have to be able to

### Magento 2 Modes

	Static File Caching	Exceptions Displayed	Exceptions Logged	Performance (-) Impacted
Developer		✓		✓
Production			✓	
Default	✓		✓	✓



### >\_ Command line

#### Cache Management

#### Commands

*Clear all caches:* php magento cache:clean

*Clearing a single cache:* php magento cache:clean config

*Flush all caches:* php magento cache:flush

*Flushing a single cache:* php magento cache:flush config

#### Index Management

#### Commands

*Runs all available indexes:* php magento indexer:re-index

#### Cron

#### Commands

*Runs all cron jobs by schedule:* php magento cron:run

#### Help & Knowledge

#### Commands

*Display help/infos about a command* php magento help indexer:re-index

*Lists all available CLI commands* php magento list

#### Development & Production

#### Commands

*Set to the production mode to harden your setup* php magento deploy:mode:setproduction

*(exceptions are not displayed to the user, only logged to log files)*

*Developer mode is less secure than Production mode* php magento deploy:mode:setdeveloper

*provides verbose logging (errors displayed in browser) enhanced debugging and disables static view file caching*

#### Recompile Store

#### Commands

Recompile the Magento 2 codebase php magento setup:di-compile

### 🔄 Useful Code Snippets and Tips

#### 📌 How to get product collection in custom template block

```
$objectManager = \Magento\Framework\App\ObjectManager::getInstance()
/** @var \Magento\Catalog\Model\ResourceModel\Product\Collection $productCollection */
$productCollection = $objectManager->create('Magento\Catalog\Model\ResourceModel\Product\Collection');
/** Apply filters here */
$productCollection->load();
```

#### 📌 How to use Mage::log method in Magento 2

```
protected $logger;
public function __construct(\Psr\Log\LoggerInterface $logger) {
    $this->logger = $logger;
}
```

### 🔄 Useful Code Snippets and Tips (cont)

You use debug, exception, system for psr logger for example

```
$this->logger->info($message);
$this->logger->debug($message);
```

#### 📌 How to retrieve product information in Magento 2

In Magento 2 proposed to use service layer for this. Try use \Magento

#### 📌 How to get a store in Magento 2 programmatically?

Magento 1.x

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

function rm_store($id = null) {
    /** @var \Magento\Framework\ObjectManager $objectManager */
    $objectManager = \Magento\Framework\App\ObjectManager::getInstance();
    /** @var \Magento_Store_Model_StoreManagerInterface $storeManagerInterface */
    $storeManagerInterface = $objectManager->get('Magento_Store_Model_StoreManagerInterface');
    return $storeManagerInterface->getStore($id);
}
```

#### 📌 How to make JavaScript strings translatable (localizable)

To make JavaScript strings translatable or localizable in Magento 2, use

*Example:* \$.mage.\_\_('Select type of option.')

Some Magento 2 core modules use another way to translate JavaScript

#### 📌 How to call theme images from a static block

You call theme images from static block as follows: `{{view url="images/`

#### 📌 How to instantiate a model in Magento 2

Magento strictly discourages the use of ObjectManager directly. It provides a factory for all non-injectables (models) you have to use factory:

```
protected $pageFactory;
public function __construct(\Magento\Cms\Model\PageFactory $pageFactory) {
    $this->pageFactory = $pageFactory;
}
public function someFunc() {
    ... $page = $this->pageFactory->create(); ...
}
```

#### 📌 How to get the collection of custom Magento 2 modules

```
protected $myModuleModelFactory;
public function __construct(... \[Namespace] \[Module] \[Model] \[Factory], ...) {
    ... $this->myModuleModelFactory = $myModuleModelFactory;
}
```

and you can use in any one of the class methods:

```
$collection = $this->myModuleModelFactory->create();
```

#### 📌 How to get POST and GET requests in Magento 2

```
public function __construct(\Psr\Log\LoggerInterface $logger) {
    $this->logger = $logger;
}
```



### 🔄 Useful Code Snippets and Tips (cont)

In a case of a controller that extends `Magento\Framework\App\Action\Action`, it is possible to get the request with the aid of `$this->getRequest()`.

For a custom class, inject the request in the constructor:

```
namespace NameSpace \Module\Something;
class ClassName {
    protected $request;
    public function __construct(
        \Magento\Framework\App\Request\Http $request,
        ...//rest of parameters here    ) {
        $this->request = $request;
        ...//rest of constructor here
    }
    public function getPost() {
        return $this->request->getPost();
    }
}
```



By **Abdellatif EL MIZEB**  
(latmiz)  
[cheatography.com/latmiz/](https://cheatography.com/latmiz/)

Published 21st February, 2017.  
Last updated 21st February, 2017.  
Page 4 of 4.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish  
Yours!  
<https://apollopad.com>