Cheatography

Magento 2 Cheat Sheet

by [deleted] via cheatography.com/34559/cs/10816/

Command line

Cache Management	Example of use
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	**Example of use**
Runs all available indexes:	php magento indexer:reindex
Cron	**Example of use**
Runs all cron jobs by schedule:	php magento cron:run
Help & Knowledge	**Example of use**
Display help/infos about a command	php magento help indexer:reindex
Lists all available CLI commands	php magento list
Developement & Production	**Example of use**
Set to the production mode to harden your setup	php magento deploy:mode:set-production

Command line (cont)

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

Developer mode is less secure than Production mode	php magento deploy:mode:set-developer

provides verbose logging (errors displayed in browser) enhanced debuggingand disables static view file caching

Recompile Store	Example of use
Recompile the Magento 2 codebase	php magento setup:di:compile

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. Your 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

name (required)	The name of the observer for the event definition
instance (required)	The fully qualified class name of the observer

EVENTS.XML FILE (cont)

disabled	Determines whether this observer is active or not. Default value is false
shared	etermines the lifestyle of the class. Default is false

Example

```
<event name="my_module_event_before"> <observer name="myObserverName" instance="MyCompany\MyModule\Observer\MyObserver" /> </event>
```

DI.XML File

What case we use di.xml ?	ϟ We can use di.xml for (rewrite) preference of a particular class.
	ϟ We can send a new or replace the existing class arguments.
	ϟ Use plugins to do some stuff before, after and around a function
	ϟ By using virtualTypes creating a sub-class of another class.

Example for Preference

```
<preference for="Magento\Customer\Api\AddressRepositoryInterface" type="Magento\Customer\Model\ResourceModel\AddressRepository" />
```

DI.XML File (cont)

Above code, When someone asks you to instantiate a Magento\Customer\Api\AddressRepositoryInterface it will instantiate a Magento\Customer\Model\ResourceModel\AddressRepository object (the type attribute).

Example for Arguments

```
<type name="Magento\Customer\Model\ResourceModel\Group" shared="false"> <arguments> <argument name="groupManagement" xsi:type="object">Magento\Customer\Api\GroupManagementInterface\Proxy</argument> </arguments> </type>
```

In the above code, We are sending object as an argument, we are saying system to insert **"Proxy"** class as an object with the name of groupManagement. Also we can use Arguments for replacing the existing argument too.

Example for Plugin

```
<type name="Magento\Customer\Model\ResourceModel\Visitor"> <plugin name="catalogLog" type="Magento\Catalog\Model\Plugin\Log" /> </type>
```

In the above code , public function clean(-$object) in visitor class is called after public function afterClean(Visitor $subject, $logResourceModel) which is in Log class.

Example for Virtual Types

<virtualType name="ourVirtualTypeName" type="Pulsestorm\TutorialVirtualType\Model\Argument1"> </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 sub-class is which dependencies are injected.

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.

Design Patterns In Magento (cont)

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.