Getting Started with Magento 2

Publisher- Silver Touch Technologies Ltd.

Editor- Deepa Ranganathan

Contents

Magento 2 has a New Look Frontend 6

Features that Make Magento 2 an Irresistible Platform 8

How to Create a Custom Log in Magento 2? 11

How to Create a Simple Module with Magento 2.0? 13

Will Dependency Injection Prove to be a Game Changer for Magento 2? 16

Disclaimer

The objective of this e-book is to guide the readers on the new features, functionalities, database logging and module creation for Magento 2.0. Silver Touch Technologies does not warrant the codes contained in the content will meet your requirements in creating a operational module or functionality.

Introduction

Magento 2, the most awaited platform version of Magento, recently released its Beta version. A lot of developers had been awaiting its release, and could not wait to get their hands dirty with this newer and better offering. Magento 2 has offering improved structure, and has aimed to give developers the flexibility when it comes to customizing the platform to suit the design needs.

Logging, customizing and tailoring only a part of the core file have all been improved for Magento 2. With this newer version, developers can conclude better performance, and improve conversions. A brief introduction into Magento2, and how it works will help fine tune your methods for the main version.

Magento 2 has a New Look Frontend

Magento 2 has definitely come out with an improved frontend. The up and coming technologies like HTML, CSS3 and JQuery have been utilized in creating the frontend for Magento2. Few elements like the layout, and file structure have been refurbished for Magento 2. All new Magento UI library has been incorporated for the functioning of this new version.

Incorporating New Theme

Let's say you want to create a new theme. You will require the theme_root_dir/theme.xml file to initiate your new theme. Apart from initializing the theme, you will also need to include the placeholder image, declare which version of the theme you are incorporating etc. below is the code that explains what all needs to be included in the initiation process. This will seem almost similar to how it works for Magento.

```
<theme xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="../../../../../lib/internal/Magento/Framework/Config/etc/them
e.xsd">
   <title>Sttl</title>
   <version>1.0.0.0</version>
   <parent>Magento/blank</parent>
   <media>
      <preview_image>media/preview.jpg</preview_image>
   </media>
</theme>
```

What's changed?

- You will see that the new version does not include the skin directory. Instead, all the different files and folders of the theme are located within the app structure

- There is a module specific view directory which includes JS, CSS, template and layout files that are specific to the particular module in concern. This way finding the files is easy

This simply means that there is a module directory and all files are included within that directory. You will not find the different layout and template directories, as in the old version, thus reducing the complications.

You can declare the theme as you used to do it in Magento's original version. In this new version the path app/design/frontend/base/default directory is not included which means the blank theme would be your starting point. This structure is suitable for minimal customizations or modifications. It is possible to cause custom development using this structure, but chances are it might affect the overall performance of your store.

Redefined Layout

In this new version, layout files are further divided into smaller parts in order to ease out maintenance. In the previous version, there were different handles for the layout which have been transformed into individual files in this version.
In this new version, it is easier to resize the images for the different devices. The layout file view.xml placed in the following path app/design/frontend/vendorName/newTheme/etc/ is responsible for resizing. Here's an example code written to resize images

```
<view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../../lib/internal/Magento/Framework/Config/etc/view.xsd">
    <vars module="Magento_Catalog">
        <var name="product_small_image_sidebar_size">100</var>
        <var name="product_base_image_size">275</var>
        <var name="product_base_image_icon_size">48</var>
        <var name="product_list_image_size">166</var>
        <var name="product_zoom_image_size">370</var>
        <var name="product_image_white_borders">0</var>
    </vars>
</view>
```

Though this has been created with the aim of simplifying the whole image resizing process for the developers, this might not work well with the responsive themes. It was easier and more flexible to resize the images using template files.
With this new version, a container wrapper has been introduced which allows you to pass the attributes like htmlTag, htmlClass and htmlID directly using the layout files. Even the move method introduced with this new version is interesting!if you want to move block 1 into block 2, the new version automatically sets block 1 the child and block 2 the destination of the child. It also includes system validation for the xml files, which includes both individual and merged layout files present within the structure.

Features that Make Magento 2 an Irresistible Platform

Magento is definitely a leading e-commerce solution which offers a boost your business. While Magento 1.x proved to be a definite hit with most business owners, it's time for a newer and better version to work in tandem with stores and uplift the businesses. Magento 2 has generated enough curiosity with respect to its features and layouts. Here are some interesting improvements in Magento 2 that need to be highlighted. These features will definitely set standards for the coming year, as far as ecommerce is concerned.

File Structure that Supports Customizations

Magento 2 has come with improved and enhanced file structure. Two major changes have been made to the file structure. Everything will now be placed under the app structure in Magento 2. Secondly, each module in your store will have an individual View directory which will contain the module specific template, layout, .js and CSS/LESS files.

With this small improvisation, customization has been made easy. You don't need to enter the Magento core to make the customizations

Magento 2 file structure includes four types of directories, which help segregate the task, thus reducing the load on core files and structure

» Primary Directories: You cannot modify this directory. The base, library and code directory are a part of this directory structure

» System Directories: DI directory, Generation directory as well as etc directory form a part of the system directories. You can change the location of system directories using the EntryPoint class as shown below

```
$result = $entrypoint -> run ('Magento\App\Http', array(
 'app_dirs' => array(
DirectoryList :: MEDIA => array(
'path' => 'pub/media.test',
'uri' => 'pub/media'
)
)
));
```

» Application Directories: The app directory, design directory, var directory, temporary directory, cache directory, log directory, session directory and systmp directory are included in the application directory

» Public Directories: pub directory, pub_lib directory, media directory, upload directory, static directory as well as pub_view_cache_directory are included in this directory

You can use the method used to change the location of system directory to change the location of the other directories as well.

Intuitive Layout for Better Interface

Container wrapper is the new intuitive concept that has replaced the original structural block concept of Magento 1.x which used to be listed in core/text_list. Container wrappers can include blocks as well as other containers and render them all. If you want to arrange the blocks to form a custom structure, the editor makes it intuitive and easy. You are giving out an interactive interface with this improvised layout.

Let's have a look at the improvised layout structure for Magento 2

Base Layout Structure

```
__app/code/<Namespace>/<Module>
  |__/view
    |__/<area>
      |__/layout
        |--<layout_file1>.xml
        |--<layout_file2>.xml
```

Theme Layout Structure

```
__app/design/<area>/<theme_path>
  |__/<Namespace>_<Module>
    |__/layout
      |--<layout1>.xml
      |--<layout2>.xml
      |__/override/base/
      |--<layout1>.xml
```

Conventions

```
<layout xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<update handle="page_one_column" />
    <referenceContainer name="content">
    <!-- ... -->
    </referenceContainer>
</layout>
```

Enhanced Performance and Scalability

Speed and performance hacks were constantly offered by the development team while working with Magento 1.x, as that was considered to be a major issue with this version. But, things have apparently changed with Magento 2.0

Let's have a look at how Magento has managed to improvise both performance and scalability

Performance

Indexing is the way Magento is able to improve the speed of the website by keeping the index tables updated, thus expediting the speed of your query performance. In Magento 2, the new default indexers which have been included have ensured that index updates are efficient and the query performance speed is enhanced.

The newer version has come with a performance test kit which allows you to create test environments and thus verify the system performance, and optimize it accordingly. Magento 2 has integrated with Varnish cache to reduce the load on the server, thus improving the speed of the website.

Scalability

With Magento 2 you can easily have multiple admin users create and modify the products/categories. Full load caching that comes with this new version helps cache the static page thus improving the speed of the website.

Preprocessing with CSS

Magento 2 uses the CSS and LESS, instead of bootstrap, in the core. The CSS files are published together with other view static files in the pub/directory in file structure. With the help of preprocessors, the preprocessing of the files in CSS occurs in Magento 2. The two preprocessors that help in Magento 2 include:

- **LESS Preprocessor:** This preprocessor works in tandem with LESS PHP adapter to enable use of LESS
- **CSS URL Resolver:** the links in CSS source files are resolved using this preprocessor

With preprocessing, you will see that customization is made easy and the overall speed and performance of the website is increased and enhanced.

Stable JQuery

In the earlier version, prototypes were used as the main JavaScript library as jQuery wasn't too stable for use then. But, things have changed with Magento 2. The community demanded jQuery and prototype has been replaced with JQuery in the new version.

Incredible User Experience

The interface for the backend has become more user friendly as compared to earlier versions. Users can now see the statistics for each store view using the scope filter. Two management functions are performed by the menu system thus designed i.e. ecommerce and system. Flat design has been adopted to display the different elements of the menu system-product, marketing, content report etc. The product manager interface has also been enhanced for Magento 2 such that you can edit product details with ease.

How to Create a Custom Log in Magento 2?

When you are developing a website, you may find the need to log the variables or the custom messages created for the website. While Magento 2 has a built-in log facility based on Monolog library, there will come a time when you need to create a custom log. You will find the default monolog along the path "MAGENTO2_ROOT/vendor/monolog"

The main log facility class defined for Magento 2 is "Magento\Framework\Logger\Monolog" which is defined along the path "MAGENTO2_ROOT/app/etc/di.xml" using the following code

```
<preference for="Psr\Log\LoggerInterface" type="Magento\Framework\Logger\Monolog" />
```

The monolog/logger class has been extended to this class

```
<?php
/**
 * Copyright © 2015 Magento. All rights reserved.
 * See COPYING.txt for license details.
 */

namespace Magento\Framework\Logger;

use Monolog\Logger;

class Monolog extends Logger
{
}
```

Using two arguments, message string and optional array parameter, you can create logs for the class defined.

Here are some examples of how you can use the two methods defined

```
$this->_logger->addDebug($message); // log location: var/log/system.log
$this->_logger->addInfo($message); // log location: var/log/exception.log
$this->_logger->addNotice($message); // log location: var/log/exception.log
$this->_logger->addError($message); // log location: var/log/exception.log
$this->_logger->critical($e); // log location: var/log/exception.log
When creating logging using Magento 1, a static method was used
Mage::logException($e);
```

In Magento 2, an instance "Magento\Framework\Logger\Monolog" alongm with a critical method will be adopted for this purpose, as shown in the code below

```
$this->_logger->critical($e);
```

How will you get the instance of Magento/Framework/Logger/Monolog in the class defined by you

```
<?php
namespace Test\Test\Model;

class Example{// instance of $e will be converted to string (magic metod __toString() will be called).

    protected $_logger;
    public function __construct(
        \Psr\Log\LoggerInterface $logger, //log injection
        array $data = []
    ) {
        $this->_logger = $logger;
        parent::__construct($data);
    }
    public function someExampleMethod() {
        /*
        some logic of method
        */
        //accessing to logger instance and calling log method
        $this->_logger->addDebug('some text or variable');
    }
}
```

You can easily achieve this using a constructor of the class defined by you, as seen in the code above.

Now you can use the instance “$this->_logger” in the class “Test\Test\Model\Example“

Let’s say you want to log in a custom file located in a custom location. You will need to create a custom log handler. There are three handlers, which you can use, namely exception, system, and debug defined along the path “MAGENTO2_ROOT/app/etc/di.xml” file

```
<type name="Magento\Framework\Logger\Monolog">
        <arguments>
            <argument name="name" xsi:type="string">main</argument>
            <argument name="handlers"  xsi:type="array">
                <item name="exception" xsi:type="object">Magento\Framework\Logger\Handler\Critical</item>
                <item name="system" xsi:type="object">Magento\Framework\Logger\Handler\System</item>
                <item name="debug" xsi:type="object">Magento\Framework\Logger\Handler\Debug</item>
            </argument>
        </arguments>
</type>
```

With this code, you know how to create the custom log to log in your custom file.

In case the log object exists, you don’t need to send the log through the constructor.

How to Create a Simple Module with Magento 2.0?

Magento 2.0 has been the talk for some time now. Developers are working through its functionalities, and creating example versions to understand the framework. Here you will learn how to create a simple module using this platform

Declare the Module

The module that you are creating is Magento_Hello, an example simple module. Before you do anything, you will need to declare the module. Begin by writing the file module.xml in the path

```
app/code/magento/hello/etc/module.xml
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../lib/internal/Magento/Framework/Module/etc/module.xsd">
  <module name="Magento_Hello" schema_version="0.0.1"/>
</config>
```

You will need to write the above mentioned code along the path mentioned to declare the module.

Configure the Module

Your next step is to configure the module, Magento_Hello that you have just created. You will need to create a controller and an action.

Create a file index.php along the following path

```
app/code/Magento/Hello/Controller/Index/Index.php
```

index, the folder is the controller while the file index.php is the action in this case. You will need to use execute () function to control the action

Here’s the code that will trigger the action for this module

```
namespace Magento\Hello\Controller\Index;
class Index extends \Magento\Framework\App\Action\Action
{
 public function execute()
{
    $this->_view->loadLayout();
      $this->_view->getLayout()->initMessages();
   $this->_view->renderLayout();
}
}
```

Create a Block

You will need to create a block in order to configure the module. Go to

```
app/code/Magento/Hello/Block/Hello.php
namespace Magento\Hello\Block;
```

Paste the following code to this path

```
class Hello extends \Magento\Framework\View\Element\Template
{
public function _prepareLayout()
{
    return parent::_prepareLayout();
}
}
```

Write the Configuration File

Here, config.xml will configure the default configuration within tag<default>

The frontend router information will be placed in the following path

Magento/Hello/etc/frontend/routes.xml

The front end event will be declared along the following path

Magento/Hello/ect/frontend/events.xml

As mentioned the routers are declared using the following code in

Magento/Hello/etc/frontend/routes.xml

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../../lib/internal/Magento/Framework/App/etc/routes.xsd">
<router id="standard">
    <route id="hello" frontName="hello">
        <module name="Magento_Hello" />
    </route>
</router>
</config>
```

Create Frontend Template

You will need to write the layout file to the following path

app\code\Magento\Hello\view\frontend\layout\hello_index_index.xml

The layout file uses the nomenclature based on its structure

router name_controlle namer_action name

Here's the code for the layout file

```
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../../../../../lib/internal/Magento/Framework/View/Layout/etc/page_config
uration.xsd">
<body>
    <referenceContainer name="content">
        <block class="Magento\Hello\Block\Hello" name="hello" template="success.phtml">
        </block>
    </referenceContainer>
</body>
</page>
```

Now, create a file success.phtml to the following path

app\code\Magento\Hello\view\frontend\templates\success.phtml

```
<?php echo 'Successful! This is a simple module in Magento 2.0  ; ?>
```

This will act as the reporting file, which will announce the successful creation of a module

Activate the Module

Finally, you will need to activate the module by opening the config file app/etc/config.xml

In the array module, add the element Magento_Hello=>1

You will need to run the following link to see your module live

http://localhost/magento20/hello/index/index

Conclusion

Creating a simple module certainly differs from Magento 1.0, but follows similar steps. You will see that creating a module is quite easy and effortless. You just need to remember the structure and should give a valid nomenclature to the module right when you are creating it.

Will Dependency Injection Prove to be a Game Changer for Magento 2?

Magento 2 has revamped a lot of baggage that comes from Magento 1, and the biggest change that developers need to focus on is Dependency Injection. It not only differentiates Magento 2 from its predecessors, but also from other platforms. It is definitely going to change how class and objects are created.

Dependency injection is a design pattern, which has been created as a substitute for Mage Class, and which will help manage dependencies. This design pattern will first speculate on the resources needed to create an object, instead of creating the object and then focusing on the resources. This design pattern ensures that all the dependents are decoupled, and they form a system only when coupled with a system.

You can cause class isolation as well as independent development and unit testing when using this design pattern. Let's try and understand this pattern with an example.

```
class A
{
 ...
 public function read()
 {
  $dbh = new DatabaseConnection(); // avoid this
  $dbh->query('SELECT ...');
  ...
 }
 ...
}
```

Note: This is not the full code, as the purpose here is to understand what dependency injection is, and not learn coding.

Here, you have create a simple class A combined with the method Read. For this code to work, you need a database connection object, and you create that as part of the code. This is how the object is created in Magent 1.x

Now, that you refreshed how objects are created in Magento 1.x, let's take a step further and understand how dependency injection can be used to simplify the process.

```
class A
{
 protected $_dbh;
 ...
 public function __construct(DatabaseConnection $connection)
 {
  $this->_dbh = $connection;
 }
 ...
 public function read()
 {
  $this->_dbh->query('SELECT ...');
  ...
 }
 ...
}
```

Here, you don't create the database connection as part of the Read() method. Instead, it is passed to class A through a constructor. It is important to note here that this is certainly not the only method to pass on dependencies, but for this example this is the method being considered.

Here, Class A is independent of Class Database Connection. In case, you want to interconnect the two when developing a system, you can easily do so. Otherwise, at all instances the two will remain independent of each other.

The other thing taken care of as part of this injection is large codes. The second part introduced a 52 line code, which is definitely going to make things difficult when you set to debug. When it comes to initialization of such long codes, the injection container or objectManager comes to your help. Here's how it works in a typical case

```
class A
{
  public function __construct(B $b)
  {
    $this->_b = $b;
  }
}
```

What happens when you create an object using class A?

- ObjectManager>Create ('A') is called
- After this function is called, constructor of A is examined thoroughly
- Finally class B is created, which will help in creating class A

In three steps object manager could create class A using class B. Now, this sounds too simple to be true, right? Let's take an instance where Class B has to be involved in multiple implementations of the same object. Which implementation would be used to complete the task? The solution lies in the file di.xml which is located along the following different paths

- {moduleDir}/etc/{areaCode}/di.xml
- {moduleDir}/etc/di.xml
- app/etc/di/*.xml

You will need to specify certain settings in these file paths

- Class Definitions: You need to specify the type and number of dependencies. With a constructor signature, you can compile this
- Instance Configuration: You will find ways to instantiate the object here. You will need to provide settings for type and virtualTypes as part of this configuration
- Abstraction-Implemetation Mapping: You can choose the interface of your preference for implementation here

Type of Objects

The types of objects being injected are segmented into two categories in Magento 2: Injectable and Non-injectable. Objects, which have a certain identity, like products, orders, cart items etc. are non-injectables. You will need to send a request to the particular factory to be able to access the non-injectables. You will use proxies if you are planning on loading expensive dependencies. You should ideally use non-injectable when loading complex items, as in this manner they will act as singletons.

Conclusion

Magento 2 has proven to be a game changer especially with the introduction of various features that will help captivate the users. The new version will offer an incredible frontend, and interface features which will help engage the users on the store.

There are some features new to Magento 2, which are unique and will help you develop your store thus offering substantial boost to your business. With the feature of dependency injection, Magento 2 is definitely offering flexibility and scalability when it comes to customization. This platform is definitely something to look forward to.

THANKS FOR READING

SILVER TOUCH®

To learn more, Visit us on the web at

www.semaphore-software.com

www.silvertouch.com/MagentoExtensions

497 Route 27, Iselin, NJ 08830, United States

Phone: + 1 201 299 3668 | Email: info@silvertouch.com