Online Merchant v2.4 Joli

Documentation for osCommerce Online Merchant v2.4 Joli.

Chapters

Copyright

Developers

Pages

Framework OSC\OM

Introduction

OSC\OM is a framework utilizing new features in PHP v5.5 to improve the performance, security, and modularity of the codebase. Taking advantage of namespaces and autoloading, it is now even easier to add new features and extend existing features without the need to edit core source code files.

The base framework is located in the *includes/OSC* directory:

Framework	Namespace	Location
Core	OSC\OM	includes/OSC/OM
Sites	OSC\Sites	includes/OSC/Sites
Apps	OSC\Apps	includes/OSC/Apps

Namespaces/Autoloader

The framework is built utilizing the PSR-4 standard for autoloading classes from matching namespaces and file paths. Classes are automatically loaded on demand and don't need to be included/required manually.

The base namespace the autoloader watches for is *OSC* and loads the class files located in the *includes/OSC/* directory.

Examples

Class	File Location
OSC\OM\OSCOM	includes/OSC/OM/OSCOM.php
OSC\OM\Db	includes/OSC/OM/Db.php
OSC\OM\Registry	includes/OSC/OM/Registry.php
OSC\Sites\Shop\Shop	includes/OSC/Sites/Shop/Shop.php
OSC\Apps\VENDOR\APP\APP	includes/OSC/Apps/VENDOR/APP/APP.php

Classes in the framework **must** declare their namespace as the first PHP code in the file.

Example:

```
<?php
namespace OSC\OM;

class NewClass
{
    ....
}
```

The full namespace to the above example would be:

```
OSC\OM\NewClass
```

and the location of the file would be:

```
includes/OSC/OM/NewClass.php
```

For another class to be able to load *OSC\OM\NewClass* automatically, it needs to be declared with PHP's *use* keyword after the namespace of the class and before any other PHP code.

Example:

```
<?php
namespace OSC\OM;

use OSC\OM\NewClass;

class AnotherNewClass
{
    public function __construct()
    {
        $NewClass = new NewClass();
    }
}
```

The framework autoloader (*OSC\OM\OSCOM::autoload()*) is registered as an autoloader in *includes/application_top.php* and is automatically initialized in all main PHP files that include the *application_top.php* file.

Template files do not need to have a namespace set, but still need to reference framework classes that it uses:

```
<?php
use OSC\OM\HTML;

echo HTML::outputProtected('Hello World!');
?>
```

More information about namespaces can be found at the PHP Namespace documentation page.

Sites

Sites are registered in the framework to initialize and apply environment parameters specific to that site.

The available sites in the core are:

Site	Controller
Shop *(default)*	OSC\Sites\Shop\Shop
Admin	OSC\Sites\Admin\Admin

Sites are registered and retrieved as follows:

```
use OSC\OM\OSCOM;

OSCOM::initialize();
OSCOM::loadSite('Shop');

$site = OSCOM::getSite();
```

Apps

Apps are self-contained packages that add new or extend existing features through modules and hooks. Apps reside in their own directory and do not need to edit core source code files.

Apps are located in the following directory:

Namespace	Location
OSC\Apps\VENDOR\APP	includes/OSC/Apps/VENDOR/APP

Apps also have a public directory for public accessible files such as stylesheets, javascript, and images, located at:

```
public/Apps/VENDOR/APP
```

More information is available in the Apps chapter.

Coding Standards

The framework is coded with the PSR-2 coding style guide.

Configuration

Introduction

The main installation configuration parameters are stored in the following locations:

Type	Location
Global	includes/OSC/Conf/global.php
Per-Site	includes/OSC/Sites/SITE/site_conf.php

The *global* configuration file and all *site* configuration files are automatically loaded into their own groups when the framework is initialized. The *global* configuration file is loaded into a *'global'* group, and the *site* configuration files are loaded into their own Site group (eg, *'Admin'*, and *'Shop'*).

Reading a configuration value is first attempted at the *Site* level, and if the configuration key does not exist, the *global* value is returned. A *Site* level configuration parameter has priority over a *global* level parameter if a *global* level configuration parameter is also defined.

Custom Configuration Files

It's possible to create custom configuration files that have priority over the values from the core configuration files. Custom configuration files can be stored in the following locations:

Type	Location
Global	includes/OSC/Custom/Conf/global.php
Per-Site	includes/OSC/Custom/Sites/SITE/site_conf.php

Configuration File Format

The format of the configuration parameters are stored in a "ini" style format in a PHP file that is assigned to a *$ini* PHP variable. This style of configuration was chosen over a plain text .ini file to prevent configuration parameters being read in cases of the configuration files being publicly accessible through the web server.

An example format for the *global* configuration file is:

```
<?php
$ini = <<<EOD
db_server = "localhost"
db_server_username = "dbuser"
db_server_password = "dbpass"
db_database = "oscommerce"
db_table_prefix = "osc_"
store_sessions = "MySQL"
time_zone = "Europe/Berlin"
EOD;
```

An example of a *Site* configuration file is:

```
<?php
$ini = <<<EOD
dir_root = "/www/html/"
http_server = "https://demo.oscommerce.shop"
http_path = "/"
http_images_path = "images/"
http_cookie_domain = ".oscommerce.shop"
http_cookie_path = "/"
EOD;
```

External Configuration Files

External configuration files can be loaded using the following code:

```
use OSC\OM\OSCOM;

OSCOM::loadConfigFile($path_of_file, 'ext_group');
```

This would load the configuration parameters of *$path_of_file* to the *'ext_group'* configuration group.

It is important that the ini format is stored as a string to the *$ini* PHP variable otherwise the configuration parameters can not be parsed.

Retrieving Configuration Parameters

Configuration parameters can be retrieved using *OSCOM::getConfig()*:

```
use OSC\OM\OSCOM;

$value = OSCOM::getConfig('cfg_name');
```

In this example, the *cfg_name* configuration parameter from the current Site group is returned. If the current Site group does not contain the configuration parameter, the *global* group value is returned.

It's possible to define which group the configuration parameter should be loaded from by defining the group name:

```
use OSC\OM\OSCOM;

$value = OSCOM::getConfig('cfg_name', 'ext_group');
```

This would load the *cfg_name* configuration parameter from the *ext_group* group.

The following can be used to first see if a configuration parameter exists:

```
use OSC\OM\OSCOM;

if (OSCOM::configExists('cfg_name')) {
    ....
}
```

This will check if *cfg_name* exists in the current Site group or the *global* group.

Checking to see if a configuration parameter exists in a specific group is performed as follows:

```
use OSC\OM\OSCOM;

if (OSCOM::configExists('cfg_name', 'ext_group')) {
    ....
}
```

Please note that if the configuration parameter does not exist in the specified group, a check is also performed in the *global* group.

Setting Configuration Parameters

Runtime configuration parameters can be set as follows:

```
use OSC\OM\OSCOM;

$value = true;

OSCOM::setConfig('is_true', $value, 'ext_group');
```

If no group is specified in the third parameter, the configuration parameter would be set in the *global* group.

This function does not save the configuration parameter to the configuration file - it only sets a runtime configuration parameter value.

Database OSC\OM\Db

Introduction

The *Db* class manages the connection to the database server and executes sql queries. It extends the native PHP PDO class with custom functionality optimized to the framework.

The class executes sql queries safely and securely by binding parameter values to the query via placeholders rather then having the values being injected into the sql query itself.

Connections

Db::initialize() opens a new connection to the database server. All parameters of the function are optional where the installation configuration values are used as default values.

```
use OSC\OM\Db;

$OSCOM_Db = Db::initialize();
```

Parameters

```
Db::initialize($server, $username, $password, $database, $port, array $driver_options)
```

Parameter	Value
$server	The address of the database server. Default: *db_server*
$username	The username to connect to the database server with. Default: *db_server_username*
$password	The password of the user account. Default: *db_server_password*
$database	The name of the database. Default: *db_database*
$port	The port number of the database server. Default: *null*
$driver_options	Additional driver options to use for the database connection. Defaults: ***PDO::ATTR_ERRMODE*** *PDO::ERRMODE_WARNING* ***PDO::ATTR_DEFAULT_FETCH_MODE*** *PDO::FETCH_ASSOC* ***PDO::ATTR_STATEMENT_CLASS*** *OSC\OM\DbStatement* ***PDO::MYSQL_ATTR_INIT_COMMAND*** *set session sql_mode="STRICT_ALL_TABLES,NO_ZERO_DATE,NO_ZERO_IN_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_*

A database connection is created on each page request and is available in the Registry as *Db*.

Queries

Prepared Statements

Queries are performed with *Db::prepare()* which securely binds values to the query using placeholders.

```
$seach = 'chocolate';
$category_id = 1;
$price = '4.99';

$Qproducts = $OSCOM_Db->prepare('select title from :table_products where description like :description and category_id = :category_id and status = :status and price < :price order by title');
$Qproducts->bindValue(':description', '%' . $chocolate . '%');
$Qproducts->bindInt(':category_id', $category_id);
$Qproducts->bindBool(':status', true);
$Qproducts->bindDecimal(':price', $price);
$Qproducts->execute();

while ($Qproducts->fetch()) {
    echo $Qproducts->value('title');
}
```

Binding Parameters

Parameters can be binded to the query using the following functions:

Value Type	Function
String	bindValue
Integer	bindInt
Boolean	bindBool
Decimal	bindDecimal
Null	bindNull

Table names prefixed with :table_ are binded and prefixed automatically with *db_table_prefix*.

Single Function Calls

Select Queries

Simple select queries that do not need parameters to be binded can be executed with *Db::query()*. This functions returns a normal result set.

```
$Qstates = $OSCOM_Db->query('select id, title from :table_states where country_id = 1 order by title');

while ($Qstates->fetch()) {
    echo $Qstates->value('title');
}
```

Update/Delete Queries

Simple update/delete queries that do not need parameters to be binded can be executed with *Db::exec()*. This functions returns the number of rows affected.

```
$result = $OSCOM_Db->exec('delete from :table_states where country_id = 1');

echo 'Affected rows: ' . $result;
```

Results

Results can be returned as a single result set, a multiple result set, and as an array containing all rows or columns.

Fetching

Single Result Set

Returning a single result set is performed as:

```
$Qstate = $OSCOM_Db->prepare('select title from :table_states where id = :id');
$Qstate->bindInt(':id', 1);
$Qstate->execute();

if ($Qstate->fetch() !== false) {
    echo 'State: ' . $Qstate->value('title');
}
```

Multiple Result Set

Returning a multiple result set is performed as:

```
$Qstates = $OSCOM_Db->prepare('select id, title from :table_states where country_id = :country_id');
$Qstates->bindInt(':country_id', 1);
$Qstates->execute();

while ($Qstates->fetch()) {
    echo 'State: ' . $Qstates->value('title');
}
```

Array Result Set

An array can be retrieved containing either all rows of the result set or all columns of the current row:

```
$Qstates = $OSCOM_Db->prepare('select id, title from :table_states where country_id = :country_id');
$Qstates->bindInt(':country_id', 1);
$Qstates->execute();

$states_all = $Qstates->fetchAll();

$current_state = $Qstates->toArray();
```

Result Exists

Checking to see if a result exists is performed as:

```
$Qstates = $OSCOM_Db->prepare('select id, title from :table_states where country_id = :country_id');
$Qstates->bindInt(':country_id', 1);
$Qstates->execute();

if ($Qstates->fetch() !== false) {
    echo 'States:';

    do {
        echo $Qstates->value('title');
    } while ($Qstates->fetch());
} else {
    echo 'No states exist.';
}
```

Please note that the following will not work:

```
$Qstates = $OSCOM_Db->prepare('select id, title from :table_states where country_id = :country_id');
$Qstates->bindInt(':country_id', 1);
$Qstates->execute();

if ($Qstates->fetch() !== false) {
    echo 'States:';

    while ($Qstates->fetch()) {
        echo $Qstates->value('title');
    }
}
```

as calling *fetch()* in the *if* statement to check if a row exists and looping through the results again in the *while* statement will skip the first row of the result set due to the first call to *fetch()*. The *do { .. } while (..)* method shown above is the correct way.

Type Hinting

Columns can be returned as a specific variable type using the following functions:

Value Type	Function
String	value
HTML Safe String	valueProtected
Integer	valueInt
Decimal	valueDecimal

```
$Qproducts = $OSCOM_Db->prepare('select id, title, code, price from :table_products where description like :description order by ti
tle');
$Qproducts->bindValue(':description', '%chocolate%');
$Qproducts->execute();

if ($Qproducts->fetch() !== false) {
    do {
        echo $Qproducts->valueInt('id') . ': ' . $Qproducts->valueProtected('title') . ' (' . $Qproducts->value('code') . ') = ' .
             $Qproducts->valueDecimal('price');
    } while ($Qproducts->fetch());
}
```

Affected Rows

The number of rows affected by an *insert*, *update*, or *delete* query can be returned as:

```
$Qupdate = $OSCOM_Db->prepare('update :table_states set title = :title where id = :id');
$Qupdate->bindValue(':title', 'Beverly Hills');
$Qupdate->bindInt(':id', 1);
$Qupdate->execute;

echo 'Affected rows: ' . $Qupdate->rowCount();
```

Please do not use *rowCount()* for *select* queries as this is not supported by PDO.

Total Rows

Retrieving the total rows of a query can be performed as:

```
$Qtotal = $OSCOM_Db->prepare('select SQL_CALC_FOUND_ROWS id from :table_orders where status = :status');
$Qtotal->bindBool(':status', true);
$Qtotal->execute();

echo 'Total rows: ' . $Qtotal->getPageSetTotalRows();
```

getPageSetTotalRows() requires *SQL_CALC_FOUND_ROWS* to exist in the query and automatically retrieves the total rows using *select found_rows()* after the query has been executed.

It is also possible to use *fetchAll()* however this method uses more server resources and is not recommended:

```
$Qorders = $OSCOM_Db->prepare('select id from :table_orders where status = :status');
$Qorders->bindBool(':status', true);
$Qorders->execute();

echo 'Total rows: ' . count($Qtotal->fetchAll());
```

Page Sets

Returning a page set result is performed as:

```
$Qorders = $OSCOM_Db->prepare('select SQL_CALC_FOUND_ROWS order_number, total_price from :table_orders where customer_id = :custome
r_id and status = :status order by id desc limit :page_set_offset, :page_set_max_results');
$Qorders->bindInt(':customer_id', 1);
$Qorders->bindBool(':status', true);
$Qorders->setPageSet(15);
$Qorders->execute();

if ($Qorders->getPageSetTotalRows() > 0) {
    echo 'Orders';

    while ($Qorders->fetch()) {
        echo 'Order #' . $Qorders->valueInt('order_number') . ': ' . $Qorders->valueDecimal('total_price');
    }

    echo $Qorders->getPageSetLabel('Displaying <strong>{{listing_from}}</strong> to <strong>{{listing_to}}</strong> (of <strong>{{l
isting_total}}</strong> orders)');

    echo $Qorders->getPageSetLinks();
}
```

Parameters

```
setPageSet($max_results, $page_set_keyword, $placeholder_offset, $placeholder_max_results)
```

Parameter	Value
$max_results	The number of results to show per page.
$page_set_keyword	The name of the parameter holding the current page value. Default: *page*
$placeholder_offset	The name of the binding placeholder used as the limit offset in the sql query. Default: *page_set_offset*
$placeholder_max_results	The name of the binding placeholder used as the limit row number in the sql query. Default: *page_set_max_results*

The parameter name of the current page value is passed as the second parameter. The default value is *page* and the value is retrieved from *$_GET['page']* if it exists.

Caching

Caching of select query result sets improves performance by storing the result of the query in a cache file and subsequently reading the cached data until the cache expiration time is reached. As soon as the cache expiration time is reached, the database is queried again and the cached information is refreshed with the new result set.

```
$Qcfg = $OSCOM_Db->prepare('select key, value from :configuration');
$Qcfg->setCache('configuration');
$Qcfg->execute();

while ($Qcfg->fetch()) {
    echo $Qcfg->value('key') . ': ' . $Qcfg->value('value');
}
```

Parameters

```
setCache($key, $expire, $cache_empty_results)
```

Parameter	Value
$key	The name of the cache block to retrieve or save.
$expire	The time in minutes the cached data should be saved for. A value of 0 keeps the cached data indefinitly until it has been manually cleared. Default: *0*
$cache_empty_results	A boolean value to cache or not cache empty result sets. Default: *false*

Shortcuts

Shortcut functions wrap *Db::prepare()* into a simpler interface to help write code faster for simpler queries.

Db::get()

Db::get() can be used to retrieve rows from a simple query.

```
$Qstates = $OSCOM_Db->get('states', [
    'id',
    'title'
], [
    'country_id' => 1
], 'title');

while ($Qstates->fetch()) {
    echo $Qstates->value('title');
}
```

Parameters

```
Db::get($table, $fields, array $where, $order, $limit, $cache, array $options)
```

Parameter	Value
$table	One (string) or more tables (array) to retrieve the rows from. Aliases may be used as: `['countries as c', 'states as s']` Table names are automatically prefixed unless the *prefix_tables* option is set as false (see the $options parameter).
$fields	One (string) or more fields (array) to retrieve. Aliases may be used as: `['c.countries_id as id', 'c.countries_title as title']`
$where	Array containing keys and values matching the column name to the condition: `['id' => 1]`
$order	One (string) or more fields (array) to sort by: `['title', 'c.date_added']`
$limit	An integer value to limit the number of rows to, or an array containing two integer values to limit the number of rows (second value) with an offset (first value): `[1, 15]`
$cache	An array consisting of the parameters (in order) sent to *setCache()*.
$options	An array containing the following options: `['prefix_tables' => true]`

A more complex multi-relationship query example can be performed as:

```
$Qproducts = $OSCOM_Db->get([
    'products p',
    'products_to_categories p2c'
], [
    'count(*) as total'
], [
    'p.products_id' => [
        'rel' => 'p2c.products_id'
    ],
    'p.products_status' => '1',
    'p2c.categories_id' => '1'
]);

$products_count = $Qproducts->valueInt('total');
```

Db::save()

Db::save() can be used to insert or update data in a table.

```
$result = $OSCOM_Db->save('states', [
    'title' => 'California'
], [
    'id' => 1
]);

echo 'Affected rows: ' . $result;
```

Parameters

```
Db::save($table, array $data, array $where_condition)
```

Parameter	Value
$table	The table to save the data to.
$data	An associative key=>value array containing the data to save in the table. The array keys must match the table field names the array value should be saved in.
$where_condition	If no condition is passed, the data is inserted into the table as a new record. If an associative $key=>$value array is passed, it is used as the where condition of the query to update the data of an existing record.

Db::delete()

Db::delete() can be used to delete a single, multiple, or all records from a table.

```
$result = $OSCOM_Db->delete('states', [
    'id' => 1
]);

echo 'Affected rows: ' . $result;
```

Parameters

```
Db::delete($table, array $where_condition)
```

Parameter	Value
$table	The table to delete the records from.
$where_condition	If no condition is passed, all records in the table are deleted. If an associative $key=>$value array is passed, it is used as the where condition of the query to delete the matching records. The array keys must match the table field names the array value is matched against.

Registry OSC\OM\Registry

Introduction

The registry is an object storage container strictly used to store object instances and access them by reference. It does not allow registered objects to be overwritten unless manually specified to do so.

Adding to the Registry

Objects can be added to the registry using *Registry::set()*.

```
use OSC\OM\Db;
use OSC\OM\Registry;

$OSCOM_Db = Db::initialize();

Registry::set('Db', $OSCOM_Db);
```

Parameters

```
Registry::set($key, $value, $force)
```

Parameter	Value
$key	The name of the object to reference in the registry.
$value	The object to store in the registry.
$force	If this is enabled, overwrite an existing object with a new value. Default: *false*

Accessing the Registry

An object can be accessed in the registry using *Registry::get()*.

```
use OSC\OM\Registry;

$OSCOM_Db = Registry::get('Db');
```

Parameters

```
Registry::get($key)
```

Parameter	Value
$key	The name of the object to access in the registry.

Checking the Registry

An existing object can be checked for in the registry using *Registry::exists()*.

```
use OSC\OM\Db;
use OSC\OM\Registry;

if (Registry::exists('Db')) {
        $OSCOM_Db = Registry::get('Db');
} else {
        $OSCOM_Db = Db::initialize();

        Registry::set('Db', $OSCOM_Db);
}
```

Parameters

```
Registry::exists($key)
```

Parameter	Value
$key	The name of the object to check in the registry for.

Apps OSC\Apps\

Introduction

Apps are self-contained packages that add new features and extend existing features without the need to edit core source code files. All files of an App are located in one main directory, with an external public directory for public resources (eg, images, javascript, stylesheets, ..).

Structure

Apps are located in the following directory:

```
includes/OSC/Apps/VENDOR/APP
```

Files that need to be publicly accessible are located at:

```
public/Apps/VENDOR/APP
```

A JSON structured *oscommerce.json* metadata file must exist in the top level App directory that describes the App and the modules it makes available. Modules **do not** need to exist in a specific directory within the package - rather the namespace is defined in the metadata file which the framework uses as a reference to find the location of the module class file.

> For security reasons, modules that are not described in the App metadata file will not be loaded by the framework.

Metadata File Example: osCommerce\Core

```
{
  "title":            "osCommerce Core Modules",
  "app":              "Core",
  "vendor":           "osCommerce",
  "version":          1.0.0,
  "req_core_version": 2.4.0,
  "license":          "MIT",
  "authors": [
    {
      "name":    "osCommerce",
      "company": "osCommerce",
      "email":   "sales@oscommerce.com",
      "website": "https://www.oscommerce.com"
    }
  ],
  "modules": {
    "Payment": {
      "COD": "Module\\Payment\\COD"
    },
    "Shipping": {
      "Flat":  "Module\\Shipping\\Flat",
      "Zones": "Module\\Shipping\\Zones"
    },
    "Hooks": {
      "Shop/Cart": {
        "AdditionalCheckoutButtons": "Module\\Hooks\\Shop\\Cart\\AdditionalCheckoutButtons"
      }
    }
  },
  "routes": {
    "Admin": "Sites\\Admin\\Pages\\Home",
    "Shop": {
      "new-offers": "Sites\\Shop\\Pages\\NewOffers",
      "account&coupons": "Sites\\Shop\\Pages\\Coupons"
    }
  }
}
```

Namespaces defined in the *modules* section are based from the Apps own namespace level - a namespace of *Module\Payment\COD* has a full namespace class of *OSC\OM\Apps\VENDOR\APP\Module\Payment\COD*.

Namespaces must also be escaped appropiately according to the JSON specification otherwise the metadata file will not be readable. This means the PHP namespace separator must consist of two backslash characters instead of one.

Metadata Schema

Parameter	Description
title	The public title of the App.
app	The internal name of the App. This must be one word and match the APP directory name of the App.
vendor	The vendor / company name of the App. This must be one word and match the VENDOR directory name of the App.

Parameter	Description
version	The version of the App. This must be in X.Y.Z format.
req_core_version	The minimum osCommerce Online Merchant version required for the App. This must be in X.y.Z format.
license	The license the App is released under. Examples: • GPL • LGPL • MIT • proprietary
authors	A list of authors containing the following for each author: • name • company • email • website
modules	A list of modules the App makes available. Each module type has its own specification - please refer to the Modules section for more information.
routes	A list of url based routes that load App Page controllers. Each Site has its own specification - please refer to the Routes section for more information.

Modules that are installed and registered in the database are not stored with their full namespace class names, rather with an alias as defined in the metadata file.

For example, the core *Cash on Delivery* payment module is stored in the *MODULE_PAYMENT_INSTALLED* configuration parameter as *osCommerce\Core\COD*. The payment class knows that a payment module is being requested and looks up the *COD* alias defined in the metadata file to retrieve the class as *Module\Payment\COD*. The full class name reference and file location would be:

Namespace	Class Location
OSC\Apps\osCommerce\Core\Module\Payment\COD	includes/OSC/Apps/osCommerce/Core/Module/Payment/COD.php

Modules

Modules that are to be made available to the framework **must be** defined in the Apps metadata file. Each module type has its own definition specification to be able to load the module class on request simply by referencing an alias.

Each module must implement a module type interface which dictates the functions the module must make available to the framework. Module Type Interfaces are located in the following namespace and directory:

Namespace	Directory
OSC\OM\Modules\	includes/OSC/OM/Modules/

Module Types and Interfaces

The following Module Types are available in the core:

Module Types	
	Metadata JSON
AdminDashboard Administration Dashboard modules.	
Interface	

Module Types

OSC\OM\Modules\AdminDashboardAbstract
OSC\OM\Modules\AdminDashboardInterface

```
"app": "Core",
"vendor": "osCommerce",
"modules": {
  "AdminDashboard": {
    "NewOrders": "Module\\Admin\\Dashboard\\NewOrders",
    "NewCustomers": "Module\\Admin\\Dashboard\\NewCustomers"
  }
}
```

Specification

```
"modules": {
  "AdminDashboard": {
    "ALIAS": "CLASS"
  }
}
```

Reference

```
VENDOR \ APP \ ALIAS

osCommerce\Core\NewOrders
osCommerce\Core\NewCustomers
```

AdminMenu

Administration Menu modules.

Interface

OSC\OM\Modules\AdminMenuInterface

Metadata JSON

```
"app": "Core",
"vendor": "osCommerce",
"modules": {
  "AdminMenu": {
    "Products": "Module\\Admin\\Menu\\Products",
    "Orders": "Module\\Admin\\Menu\\Orders"
  }
}
```

Specification

```
"modules": {
  "AdminMenu": {
    "ALIAS": "CLASS"
  }
}
```

Reference

```
VENDOR \ APP \ ALIAS

osCommerce\Core\Products
osCommerce\Core\Orders
```

Content

Metadata JSON

Module Types

Content modules.

Interface

OSC\OM\Modules\ContentInterface

```
"app": "Core",
"vendor": "osCommerce",
"modules": {
  "Content": {
    "login": {
      "Social": "Module\\Content\\SocialLogin"
    }
  }
}
```

Specification

```
"modules": {
  "Content": {
    "CONTENT MODULE GROUP": {
      "ALIAS": "CLASS"
    }
  }
}
```

Reference

```
CONTENT MODULE GROUP / VENDOR \ APP \ ALIAS

login/osCommerce\Core\Social
```

Hooks

Hook modules.

Interface

OSC\OM\Modules\HooksInterface

Metadata JSON

```
"app": "Core",
"vendor": "osCommerce",
"modules": {
  "Hooks": {
    "Shop/Account": {
      "Logout": "Module\\Hooks\\Shop\\Account\Logout"
    }
  }
}
```

Specification

```
"modules": {
  "Hooks": {
    "SITE / GROUP": {
      "ACTION": "CLASS"
    }
  }
}
```

Reference

Module Types

```
VENDOR \ APP \ SITE / GROUP \ ACTION

osCommerce\Core\Shop/Account\Logout
```

Payment
Payment modules.

Interface
OSC\OM\Modules\PaymentInterface

Metadata JSON

```
"app": "Core",
"vendor": "osCommerce",
"modules": {
  "Payment": {
    "COD": "Module\\Payment\\COD"
  }
}
```

Specification

```
"modules": {
  "Payment": {
    "ALIAS": "CLASS"
  }
}
```

Reference

```
VENDOR \ APP \ ALIAS

osCommerce\Core\COD
```

Getting Class Names

The full namespace class name of a module can be retrieved via its alias using *Apps::getModuleClass()*:

```
use OSC\OM\Apps;

$class = Apps::getModuleClass('osCommerce\Core\COD', 'Payment');

// $class = OSC\Apps\osCommerce\Core\Module\Payment\COD
```

Parameters

```
Apps::getModuleClass($module, $type)
```

Parameter	Value
$module	The alias of the module to retrieve the class name from.
$type	The module interface type.

Getting Modules

A list of available App modules can be retrieved using *Apps::getModules()*:

```php
use OSC\OM\Apps;

$modules = Apps::getModules('Payment');
```

Parameters

```
Apps::getModules($type, $filter_vendor_app, $filter)
```

Parameter	Value
$type	The module type to retrieve.
$filter_vendor_app	If an App is provided, only modules from that App are returned, otherwise all modules from all Apps are returned. This can be limited to VENDOR for all vendor Apps, or VENDOR\APP for a specific App.
$filter	Pass a Module Type specific filter to filter the results with.

Routes

Routes allow Apps to load Page controllers via the Sites route resolver method. The *Shop* Site allows Apps to load Page controllers depending on the url of the page requested, and the *Admin* Site allows one main Page controller to be loaded to administrate and configure the App on the Administration Dashboard.

```
"app": "Core",
"vendor": "osCommerce",
"routes": {
  "Admin": "Sites\\Admin\\Pages\\Home",
  "Shop": {
    "new-offers": "Sites\\Shop\\Pages\\NewOffers",
    "account&coupons": "Sites\\Shop\\Pages\\Coupons"
  }
}
```

Hooks OSC\OM\Hooks

Introduction

Hooks allow action callouts to be thrown during an event to execute additional functionality. Hooks are not modules in the traditional sense of being able to be installed and configured; they are simply modular functions waiting to be executed on demand with no configuration or administration whatsoever.

For security reasons, hooks must be defined in their Apps metadata file otherwise they will not be made available for the framework to use.

Hooks are initialized by creating an instance of *OSC\OM\Hooks* and specifying the *Site* to call the hook action from. If no *Site* is passed, the *Site* that initialized the framework is used as default.

```
use OSC\OM\Hooks;

$OSCOM_Hooks = new Hooks();
```

Hooks() is automatically initialized on each page request and is available in the Registry as *Hooks*.

Action Callout

Throwing out an action callout is performed by specifying the group the action belongs to, the actual action name, and optionally a specific function to execute. There are two types of callouts that can be peformed:

Type	Description
call()	Executes the hooks and can return an array of results depending on the action.
output()	Executes the hooks and returns the output as a string.

Hooks::call()

```
// no output expected
$OSCOM_Hooks->call('Account', 'Logout');
```

Parameters

```
Hooks::call($group, $hook, $parameters, $action)
```

Parameter	Value
$group	The group the action call belongs to.
$hook	The name of the action to call.
$parameters	Any parameters to pass to the hooks.
$action	The name of the hook function to execute. Default: *execute*

Hooks::output()

```
// concatenated string output of all hooks executed
echo $OSCOM_Hooks->output('Orders', 'Page', null, 'display')
```

Parameters

```
Hooks::output($group, $hook, $parameters, $action)
```

The *output()* method shares the same function parameters as the *call()* class method.

A list of hook action callouts is available on the List of Hook Callouts page.

Runtime Watches

Runtime watches can be defined that execute when the hook is called. This differs from hooks defined in App metadata files as they are defined in-line and execute either an existing function or are defined with an anonymous function to execute:

```php
use OSC\OM\Registry;

$OSCOM_Hooks = Registry::get('Hooks');

$OSCOM_Hooks->watch('Session', 'Recreated', 'execute', function($parameters) {
    ....
});
```

Roadmap

v2.4.0 (Beta 1)

OSC\OM Framework
: Introduce a framework to replace legacy classes and functions with.

Bootstrap Frontend Framework
: Replace the jQuery UI based Shop, Admin, and Setup frontend with Bootstrap.

Self-Contained Apps
: Introduce self-contained Apps with support for self-defined modules and url routes.

Hooks
: Introduce hook calls.

Online Updates
: Introduce an online update feature for the core.

INI Style Language Definitions
: Move the language definitions to ini style files.

Upcoming

Legacy Cleanup
: Move core modules to an osCommerce App.

  Migrate remaining classes to the new framework, including:

  - Breadcrumb
  - Currencies
  - Language
  - Message Stack
  - Navigation History
  - Shopping Cart

SEO
: Introduce SEO features.

Database Schema Comparison
: Introduce a database schema comparison tool existing store owners can use as part of their upgrade.

Hook Action Calls
: Finalize hook action calls throughout the codebase.

Online Updates
: Extend the online update feature to installed Apps.

App Browser
: Introduce an online App browser and installer.

Language Packs
: Support community maintained language packs and provide a list of ready to be used languages in the setup routine.

Legacy Compatibility Module
: Automatically re-map legacy v2.3 classes, functions, and code to the new framework for compatibility with existing v2.3 add-ons.

Legacy Compatibility

This document is work in progress.

A legacy compatibility module will be made available that automatically re-maps legacy v2.3 functions and classes to the new framework to load a compatiblity layer existing v2.3 add-ons can use.

Super Globals and Session Variables

The usage of legacy PHP long variables has been replaced with PHP super globals and are no longer referenced as globals in functions.

Legacy	Super Global
$HTTP_GET_VARS	$_GET
$HTTP_POST_VARS	$_POST
$HTTP_COOKIES_VARS	$_COOKIES
$HTTP_FILES_VARS	$_FILES
$HTTP_SERVER_VARS	$_SERVER
(session variables)	$_SESSION

Database

The legacy *tep_db_** database functions have been replaced with the *OSC\OM\Db* class.

List of Hook Callouts

Site / Filename	Group	Action
-- Admin --		
categories.php	Products	PreAction ActionDelete ActionMove ActionSave ActionCopy Page
login.php	Account	LogoutBefore LogoutAfter
orders.php	Orders	PreAction Action Page
-- Shop --		
includes/OSC/OM/SessionAbstract.php	Session	StartBefore StartAfter Recreated
logoff.php	Account	Logout
shopping_cart.php	Cart	AdditionalCheckoutButtons