Controllers within PrestaShop

Controllers within PrestaShop

In an MVC architecture, a Controller manages the synchronization events between the View and the Model, and keeps them up to date. It receives all the user events and triggers the actions to perform.

If an action needs data to be changed, the Controller will “ask” the Model to change the data, and in turn the Model will notify the View that the data has been changed, so that the View can update itself.

All of PrestaShop’s controllers actually override the Controller class through another inheriting class:

- `AdminController`,
- `ModuleAdminController`,
- `FrontController`,
- `ModuleFrontController`.

They can be found in the /classes/controller folder.

The FrontController class

Here are some of the class’ properties:

Property	Description
$template	Template name for page content.
$css_files	Array list of CSS files.
$js_files	Array list of JavaScript files.
$errors	Array of errors that have occurred.
$guestAllow ed	Whether a customer who has signed out can access the page.
$initialize d	Whether the `init()` function has been called.
$iso	The ISO code of the currently selected language.
$n	The number of items per page.
$orderBy	The field used to sort.
$orderWay	Whether to sort is ascending or descending (“ASC” or “DESC”).
$p	The current page number.
$ajax	If the ajax parameter is detected in request, set this flag to true.

Execution order of the controller’s functions

1. `__contruct()`: Sets all the controller’s member variables.
2. `init()`: Initializes the controller.
3. `setMedia()` or `setMobileMedia()`: Adds all JavaScript and CSS specifics to the page so that they can be combined, compressed and cached (see PrestaShop’s CCC tool, in the back office “Performance” page, under # the “Advanced preferences” menu).
4. `postProcess()`: Handles `ajaxProcess`.
5. `initHeader()`: Called before `initContent()`.
6. `initContent()`: Initializes the content.
7. `initFooter()`: Called after `initContent()`.
8. `display()` or `displayAjax()`: Displays the content.

Existing front office controllers

Here are the default controllers, and the theme files that use them.

Controller's filename	Description
AddressController.php	Used by address.php to edit a customer's address.
AddressesController.php	Used by addresses.php to get customer's addresses.
AttachmentController.php	
AuthController.php	Used by authentication.php for customer login.
BestSalesController.php	Used by best-sales.php to get best-sellers.
CartController.php	Used by cart.php to manage the customer's cart.
CategoryController	Used by category.php to get product categories.
ChangeCurrencyController.php	
CmsController.php	Used by cms.php to get a CMS page.
CompareController.php	Used by products-comparison.php to compare products.
ContactController.php	Used by contact-form.php to send messages.
DiscountController.php	Used by discount.php to get a customer's vouchers.
GetFileController.php	
GuestTrackingController.php	Used by guest-tracking .php to manage guest orders.
HistoryController.php	Used by history.php to get a customer's orders.
IdentityController.php	Used by identity.php for customer's personal info.
IndexController.php	Used by index.php to display the homepage.
ManufacturerController.php	Used by manufacturer.php to get manufacturers.
MyAccountController.php	Used by my-account.php to manage customer account.
NewProductsController.php	Used by new-products.p hp to get new products.
OrderConfirmationController.php	Used by order-confirmation.php for order confirmation.
OrderController.php	Used by order.php to manage the five-step checkout.
OrderDetailController.php	Used by order-detail.php to get a customer order.
OrderFollowController.php	Used by order-follow.php to get a customer's returns.
OrderOpcController.php	Used by order-opc.php to manage one-page checkout.
OrderReturnController.php	Used by order-return.php to get a merchandise return.
OrderSlipController.php	Used by order-slip.php to get a customer's credit slips.
PageNotFoundController.php	Used by 404.php to manage the "Page not found" page.
ParentOrderController.php	Manages shared order code.
PasswordController.php	Used by password.php to reset a lost password.
PdfInvoiceController.php	
PdfOrderReturnController.php	
PdfOrderSlipController.php	
PricesDropController.php	Used by prices-drop.php to get discounted products.
ProductController.php	Used by product.php to get a product.
SearchController.php	Used by search.php to get search results.
SitemapController.php	Used by sitemap.php to get the sitemap.
StatisticsController.php	
StoresController.php	Used by stores.php to get store information.

SupplierController.php	Used by supplier.php to get suppliers.

Overriding a controller

Thanks to object inheritance, you can change a controller's behaviors, or add new ones.

Keep overrides for your own shop

Overrides in PrestaShop are exclusive. This means that if your module overrides one of PrestaShop's behaviors, another module will not be able to use that behavior properly, or override it in an predictable way.

Therefore, overrides should only be used for your own local modules, when you have a specific need that cannot be applied with it.

It is not recommended to use an override in a module that you intend to distribute (for instance through the PrestaShop Addons marketplace), and they are forbidden in partner modules.

How to

PrestaShop's controllers are all stored in the `/controllers` folder, and use the "Core" suffix.

For instance, when working with the Category controller:

- File: `/controllers/CategoryController.php`
- Class: `CategoryControllerCore`

In order to override a controller, you must first create a new class without the "Core" suffix, and place its file in the `/override/controllers` folder.

For instance, when overriding the Category controller:

- File: `/override/controllers/front/CategoryController.php`
- Class: `CategoryController`