Controllers: module views

Where a framework has controllers and actions, Exponential has module views - a view script or an Exponential\Runnable\ModuleView class that receives the address parameters and returns a result array - how to write one...

Source file: `doc/eng-us/help/web-framework-developers/controllers-as-module-views.md` Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/web-framework-developers/controllers-as-module-views.md

This page is for developers who write controllers. In Exponential the unit that handles one address is a **view** of a module. Classic views are PHP scripts that the kernel includes with `$Params` and `$Module` set; since 6.0.15 a view is best written as a class extending `Exponential\Runnable\ModuleView`, with a one-line script that calls it (Commands, cronjob parts and module views as classes).

Contents

1. Controller and view side by side
2. The script form
3. The class form
4. Keeping it thin
5. Before and after the view
6. Changing a kernel view

Controller and view side by side

Framework controller	Exponential module view
A class with action methods	one view per address; a class per view in the class form
Route parameters as arguments	`$Params['ItemID']` (ordered and unordered parameters)
`Request` object injected	`eZHTTPTool::instance()` for GET, POST and session variables; module actions for buttons
Returns a `Response`	returns a `$Result` array; the kernel puts `content` into the pagelayout
Authorization attribute or middleware	`'functions' => array( 'read' )` in `module.php`, checked before the view runs
Template rendering helper	`eZTemplate::factory()`, `setVariable()`, `fetch( 'design:...' )`

The script form

PHP

```php
<?php
// extension/expdemo/modules/expdemo/item.php
$Module = $Params['Module'];
$item = expDemoItem::fetch( (int)$Params['ItemID'] );
if ( !$item )
    return $Module->handleError( eZError::KERNEL_NOT_FOUND, 'kernel' );

$tpl = eZTemplate::factory();
$tpl->setVariable( 'item', $item );

$Result = array();
$Result['content'] = $tpl->fetch( 'design:expdemo/item.tpl' );
$Result['path'] = array( array( 'text' => 'Items', 'url' => 'expdemo/list' ),
                         array( 'text' => $item->attribute( 'title' ), 'url' => false ) );
```

`expDemoItem::fetch()` stands for a static method of your own persistent object class (ORM: eZPersistentObject). The result keys are explained on Responses, redirects, errors and JSON.

The class form

From A module view:

PHP

```php
<?php
// extension/myext/classes/runnable/views/report/show.php
namespace Exponential\View\Extension\Myext\Report;

class Show extends \Exponential\Runnable\ModuleView
{
    public function run( array $scope )
    {
        $Module = $scope['Params']['Module'];
        $tpl = \eZTemplate::factory();
        $tpl->setVariable( 'report', \myReportService::build( $scope['Params']['Month'] ) );
        return array( 'content' => $tpl->fetch( 'design:report/show.tpl' ),
                      'path' => array( array( 'text' => 'Report', 'url' => false ) ) );
    }
}
```

PHP

```php
<?php
// extension/myext/modules/report/show.php (named in module.php ViewList as before)
return \Exponential\View\Extension\Myext\Report\Show::main( __FILE__, get_defined_vars() );
```

The class can be unit tested, wrapped by events and re-implemented by settings. Remember to regenerate the extension autoloads after adding the class (`php bin/php/ezpgenerateautoloads.php -e`).

Keeping it thin

The kernel itself puts work that a view, a command and a cronjob part share into service classes, for example `Exponential\Service\Trash` for the trash view, `bin/php/trashpurge.php` and the `trashpurge` cronjob part (Services). Do the same: parse and check input in the view, call a plain class for the work, render the result. The plain class is what you test without a database (Testing).

Before and after the view

Every class-based view runs between two events: `runnable/view/before` (notify) and `runnable/view/after` (a filter that receives and must return the result). Register listeners in `site.ini [RunnableSettings] Listeners[]`. That is the nearest thing to controller-level interceptors such as Spring's `HandlerInterceptor` or a Symfony `kernel.controller` listener (Events and listeners).

Changing a kernel view

Need	Mechanism
Change one kernel view's behavior	`site.ini [RunnableSettings] Implementation[<view class>]=<your subclass>`; the subclass must extend the class it replaces
Replace the views or templates of a whole kernel module	an extension module override (Extension module override)
Only change the markup	a template override, no PHP at all (Template resolution)

INI

```ini
# settings/override/site.ini.append.php, or an extension's settings/site.ini.append.php
[RunnableSettings]
Implementation[Exponential\View\Kernel\Content\History]=myHistoryView
```

No kernel file is copied, so the next release updates the original underneath your subclass.

Related pages

- Custom modules and views
- Routing: modules, views and parameters
- Module view templates
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

> **Become an Exponential Partner**
>
> Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.
>
> Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.