Module views as classes

How module views, commands and cronjob parts are written as classes since Exponential 6.0.15 - the Runnable hierarchy, the one-line entry files, the scope a view receives, services shared by views and commands, the...

Source file: doc/eng-us/help/php-developers/module-views-as-classes.md Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/php-developers/module-views-as-classes.md

This page is for PHP developers who write views, commands or cronjob parts, or change kernel ones. From Exponential 6.0.15 the code of every kernel view, command and cronjob part lives in a class; the file you know keeps its path and calls that class. The full reference is Commands, cronjob parts and module views as classes.

Contents

1. Why classes
2. The hierarchy
3. Where the classes live
4. The scope a view receives
5. Services: one implementation for three entry points
6. Events around run()
7. Re-implementing a kernel view
8. Calling a runnable from code
9. The old way still works

Why classes

A plain view file cannot be extended, called or tested on its own. A class can: a site or an extension can replace one method of a kernel view through a setting, a command and a view can share a service, and a test can run the class. That is why new code is written as classes, and why every kernel view, command and cronjob part was moved into one.

The hierarchy

```
Exponential\Runnable\Runnable            kernel/private/classes/runnable/runnable.php
  Exponential\Runnable\Command           a command line script      run()
  Exponential\Runnable\CronjobPart       a cronjob part             run( array $scope )
  Exponential\Runnable\ModuleView        a module view              run( array $scope )
```

`ModuleView::main( $scriptFile, array $scope )` creates the instance (`Runnable::create()`, which honors re-implementations), runs `run()` between the events, and returns the view's result: `$Result` when the view set it, else what `run()` returned (`ModuleView::viewResult()`).

Where the classes live

Kind	Entry file	Class
kernel view	`kernel/<module>/<view>.php`	`Exponential\View\Kernel\<Module>\<View>` in `kernel/private/classes/views/<module>/<view>.php`
kernel command	`bin/php/<name>.php`	`Exponential\Command\Kernel\<Name>` in `kernel/private/classes/commands/<name>.php`
kernel cronjob part	`cronjobs/<name>.php`	`Exponential\Cronjob\Kernel\<Name>` in `kernel/private/classes/cronjobs/<name>.php`
extension view	`extension/<ext>/modules/<module>/<view>.php`	`Exponential\View\Extension\<Ext>\<Module>\<View>` in `extension/<ext>/classes/runnable/views/<module>/`
extension command	`extension/<ext>/bin/php/<name>.php`	`Exponential\Command\Extension\<Ext>\<Name>` in `extension/<ext>/classes/runnable/commands/`
extension cronjob part	`extension/<ext>/cronjobs/<name>.php`	`Exponential\Cronjob\Extension\<Ext>\<Name>` in `extension/<ext>/classes/runnable/cronjobs/`

A view whose name is a PHP reserved word gets a suffix: `kernel/section/list.php` calls `Exponential\View\Kernel\Section\ListView`.

The scope a view receives

The entry file hands over everything `eZProcess::runFile()` defined:

PHP

```php
<?php
return \Exponential\View\Kernel\Content\History::main( __FILE__, get_defined_vars() );
```

In `$scope`	**Holds**
`$scope['Params']`	the view's parameters; the module is `$scope['Params']['Module']`
`$scope['Result']`	the `$Result` variable, empty at the start
the view's parameters as own keys	only when `module.php` sets `'variable_params' => true`

Read the module as `$scope['Params']['Module']`, which every module has. Inside the class, `$this->scriptFile()` and `$this->scriptDir()` give what `__FILE__` and `__DIR__` meant in the entry file. Class names inside a namespace are written fully qualified (`\eZTemplate`, `\eZINI`).

Services: one implementation for three entry points

Where a view and a command or cronjob part do the same work, the work lives in one service class that all of them call. Examples from the kernel:

Work	Service	Called by
emptying the trash	`Exponential\Service\Trash`	`content/trash`, `bin/php/trashpurge.php`, `cronjobs/trashpurge.php`

Work	Service	Called by
removing expired sessions	`Exponential\Service\SessionGarbageCollector`	`setup/session`, `bin/php/ezsessiongc.php`, `cronjobs/session_gc.php`
clearing caches	`expCacheManager`	`setup/cache`, `bin/php/cache.php` (`exp:cache`), the cache toolbar
removing old drafts	`Exponential\Service\DraftsCleanup`	`cronjobs/old_drafts_cleanup.php`, `cronjobs/internal_drafts_cleanup.php`

Do the same in your extension: put the work into a plain class with values in and values out, and keep the view, the command and the cronjob part thin. The service is the part you test (PHPUnit without a database).

Events around run()

`main()` notifies `runnable/<kind>/before` (with `$runnable, $class, $scope`) and filters the result through `runnable/<kind>/after` (with `$result, $runnable, $class, $scope`), `<kind>` being `command`, `cronjob` or `view`. Register listeners in `site.ini`:

INI

```
[RunnableSettings]
Listeners[]=runnable/view/before@acmeViewTimer::before
Listeners[]=runnable/view/after@acmeViewTimer::after
```

An `after` listener is a filter: it must return the result, or the view loses it. `[RunnableSettings] Listeners[]` are attached in every context, the command line included; `[Event] Listeners[]` only for web requests (Events, listeners and filters).

Re-implementing a kernel view

Instead of copying a kernel view file, subclass its class and register the subclass:

INI

```
[RunnableSettings]
Implementation[Exponential\View\Kernel\Content\History]=acmeHistoryView
```

PHP

```
class acmeHistoryView extends \Exponential\View\Kernel\Content\History
{
    public function run( array $scope )
    {
        $result = parent::run( $scope );
        // change $result['content'] or $result['path'] here
        return $result;
    }
}
```

The replacement must extend the class it replaces; an entry naming anything else is ignored. No kernel file is copied, so the next release updates the original underneath your subclass. For commands, `Implementation[]` and `Listeners[]` are read from `settings/site.ini` and `settings/override/site.ini.append.php` only, because a command is created before its script reads the other settings. **Setup > RAD** and its survey list every runnable and every `Implementation[]` entry that cannot work.

Calling a runnable from code

PHP

```php
$result = \Exponential\View\Kernel\Content\History::create( 'kernel/content/history.php' )
            ->run( array( 'Params' => $params, 'Module' => $module ) );
```

`create()` honors `Implementation[]`; calling `run()` directly does not fire the events (`main()` does).

The old way still works

A view, command or cronjob part written as a plain PHP file runs exactly as before; nothing requires the class form. Write new code as classes, and move old code when you change it anyway.

Related pages

- Write your first module view
- Write a console command
- Write a cronjob part
- 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.