Kernel, lib and extensions: the layering

The dependency rules between lib/, kernel/classes, kernel/private, the module controllers, the designs and extensions - which layer may call which, where the rules are not met yet, how names show which layer code...

Source file: `doc/eng-us/help/software-engineers/kernel-lib-and-extensions.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/software-engineers/kernel-lib-and-extensions.md

This page is for engineers who decide where code goes and review whether a change respects the boundaries. The rules come from Layering and naming rules, which also lists every known violation.

Contents

1. The layers and the direction of calls
2. What each layer may do
3. Where the rules are not met
4. Names tell you the layer
5. Where a new piece of code goes

The layers and the direction of calls

```
 extension/<name>/          plugins: settings, modules, designs, classes, cronjobs
 design/<design>/templates  presentation: reads data through fetch functions and operators only
 kernel/<module>/           controllers: module.php, view entry files, function_definition.php
 kernel/private/            internal API: ezpKernel, views, commands, cronjobs, services, REST, e
zpEvent
 kernel/classes/            domain: content, users, roles, datatypes, workflows, caches, persiste
nce
 lib/                       generic libraries: eZDB, eZINI, eZTemplate, eZFile, eZSys, eZModule,
sessions
```

A layer may call the layers below it. Calls upward are exceptions, and new ones should not be added.

What each layer may do

Layer	May	Must not
`lib/`	provide infrastructure that does not know about content	depend on a module or a design; new code should not depend on `kernel/`
`kernel/classes/`	model content, users, roles, datatypes, workflows, caches	render pages or read request input; it takes values and returns objects
`kernel/<module>/` and view classes	read input through `eZHTTPTool`, call the domain, render through `eZTemplate`	print HTML itself, except downloads and JSON

Layer	May	Must not
`kernel/private/`	own the kernel end to end: the kernel handler, views, commands, cronjob parts, services, REST	be treated as a stable extension API; its signatures may change
`design/`	present data, escape output with `|wash`	query the database; contain inline CSS or inline script in the administration designs
`extension/`	use everything below through documented extension points	require changes to kernel files

Work that a view, a command and a cronjob part share goes into a service class in `kernel/private/classes/services/` (namespace `Exponential\Service`), not into one of them (Commands, cronjob parts and module views as classes).

Where the rules are not met

The standard records these openly; treat them as debt, not as precedent:

- Several files in `lib/` call kernel classes (`eZUser`, `eZClusterFileHandler`, `eZPersistentObject`). Treat `lib/` as "lower level than the kernel", not as an independent library set.
- A small number of view classes still read `$_POST` or `$_GET` directly instead of `eZHTTPTool`.
- `ezpEvent` is documented as private API, yet extensions rely on it.
- Plain-file views, cronjob parts and commands still run beside the class-based form.

See Handling technical debt for how such items are recorded and reduced.

Names tell you the layer

Name	Layer
`eZ*`, `ezp*` (`eZContentObject`, `ezpKernelWeb`)	inherited kernel and library code; never renamed
`exp*` (`expContentSecurityPolicy`, `expAuditConfig`)	code added since the product was renamed; new kernel code uses exp
`Exponential\View\Kernel\...`, `Exponential\Command\Kernel\...`, `Exponential\Cronjob\Kernel\...`	class-based views, commands and cronjob parts of the kernel
`Exponential\Command\Extension\<Ext>`, `Exponential\View\Extension\<Ext>\<Module>`	the same for an extension
`sevenx*`	7x's public PHP prefix, used by some extensions (The prefix rule)

One class per file, and the file name is the class name in lower case: `expContentSecurityPolicy` lives in `expcontentsecuritypolicy.php`. After adding or renaming a class, regenerate the class maps (`php bin/php/ezpgenerateautoloads.php -e`; `-k` for the kernel).

Where a new piece of code goes

You are writing	Put it in
A feature for one installation or a reusable plugin	an extension: `extension/<ext>/` (Anatomy of an extension)
A new kernel subsystem	`kernel/classes/<subsystem>/` for the domain, `kernel/private/classes/` for views, commands and services
Infrastructure without content knowledge	`lib/`, without new kernel dependencies
A page	a module view: a `ModuleView` subclass plus its template
A setting	its default with a comment in `settings/<file>.ini`, read through `eZINI`
Text shown to users	`ezpI18n::tr( $context, $text )` in PHP, `i18n( $context )` in templates

Related pages

- Architecture overview
- Events, extension points and plugins
- Extension points at a glance
- 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.