Extension points from PHP

Where PHP code plugs into Exponential without editing the kernel - the registration mechanisms (directory, handler, autoload, file, design, ini, registry, filter, override), the extension points a PHP developer uses...

Source file: `doc/eng-us/help/php-developers/extension-points-from-php.md` Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/php-developers/extension-points-from-php.md

This page is for PHP developers who want to change or add behavior and look for the right place to plug in. Exponential is extended in a handful of ways that repeat. `kernel/setup/expradcatalogue.php` (`expRADCatalogue::points()`) lists every extension point, and **Setup > RAD** (`/setup/rad`) is drawn from it: 70 extension points, 65 of them with a tool. The full list with each INI line is Extension points.

Contents

1. The mechanisms
2. The points PHP developers use most
3. Choosing the lightest mechanism
4. Handlers named by settings
5. The RAD pages and the survey

The mechanisms

Mechanism	How it is found	Example
directory	a class in a directory the kernel scans, named by a `...Directories` setting	datatypes, workflow event types
handler	a class named by a setting and loaded through `eZExtension::getHandlerClass()`	session handler, cluster file handler, mail transport
autoload	a class reached through an extension autoload path	template operators (`ExtensionAutoloadPath[]`)
file	a file in a place the kernel looks by name	`function_definition.php`, `module.php`
design	a template in a design, found through the design chain	override templates
ini	settings only	most behavior switches
registry	entries of a registry that `rad.ini` declares	debug bar settings, system card providers
filter	a listener of a filter event in `site.ini [Event] Listeners[]`	`response/output`, `httpcache/store/headers`
override	a kernel class replaced through `var/autoload/ezp_override.php`	the last resort

The points PHP developers use most

You want to	Point	Page
add a field type	datatype	Write a datatype
add a workflow step	workflow event type	Write a workflow event type
add pages	module and views	Write your first module view
change one kernel view, command or cronjob part	`[RunnableSettings] Implementation[]` subclass	Module views as classes
give templates data	fetch function	Write a fetch function
give templates a helper	template operator	Write a template operator in PHP
react to the kernel	kernel event listener	Events, listeners and filters
change a value on its way	filter event listener	Events, listeners and filters
JSON for your own JavaScript	server-side ajax function	ezjscore JSON endpoints in PHP
new REST routes	REST provider	REST and remote services from PHP
add a policy with limitations	policy function and limitation	Policy functions and limitations
scheduled work	cronjob script and part	Write a cronjob part
a command	extension command	Write a console command

Choosing the lightest mechanism

Prefer, in this order:

1. **Settings** (`ini`), when a setting already does what you need.
2. **A template override** (`design`), when only output changes.
3. **A registered point** (directory, handler, autoload, file, registry): the kernel calls your class by name, and several extensions can live side by side.
4. **A listener or filter**, when you react to or adjust what the kernel does.
5. **A `[RunnableSettings] Implementation[]` subclass**, when one method of a kernel view, command or cronjob part must change.
6. **A kernel override**, only when nothing else works. Nothing registers the replacement by name, two extensions replacing the same class conflict silently, and you must follow every kernel update by hand (Kernel overrides, the last resort).

Handlers named by settings

Many infrastructure parts (session, cluster files, mail transport, search engine, image handler, static cache) are classes named by a setting, loaded with `eZExtension::getHandlerClass()`. Your class implements the same interface or extends the kernel's class, and the setting names

it. A handler named in a setting must exist before the setting goes live; otherwise every request that needs it fails. Deploy code first, settings second, and guard with `class_exists()` where a missing class must not break pages (Handlers named by settings).

The RAD pages and the survey

- `/setup/rad` shows every extension point, its mechanism, the setting that registers it and a tool that writes a skeleton for it (datatype, template operator, workflow event, module, settings extension ...) (RAD tools, RAD tools feature).
- `/setup/radsurvey` counts what your installation registers at each point, the runnables and their re-implementations, and every entry that cannot work (a class that does not exist, a replacement that does not extend the class it replaces).

Related pages

- Extension points at a glance
- RAD extension surface
- Extending without forking
- 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.