How a request runs in PHP

The path of one web request through Exponential's PHP code - index.php, autoload.php, ezpKernel and ezpKernelWeb, request initialization, URL translation, access checks, the module view, the pagelayout and shutdown...

Source file: `doc/eng-us/help/php-developers/how-a-request-runs-in-php.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/php-developers/how-a-request-runs-in-php.md

This page is for PHP developers who want to know what happens between the web server handing a request to PHP and the page leaving it. Knowing the order tells you where your code runs, what is already set up when it runs, and which events you can hook into. The names below are the real classes and methods; open the files next to this page.

Contents

1. The short version
2. index.php
3. autoload.php
4. ezpKernelWeb: initialization
5. The dispatch loop
6. The pagelayout and output
7. Shutdown
8. Events on the way
9. The command line is different

The short version

```
index.php
  -> autoload.php                    EXP_ROOT_DIR, phpcompat on PHP < 8.1, phar decision, class m
ap autoloader
  -> new ezpKernel( new ezpKernelWeb() )->run()
       ezpKernelWeb::run()
         requestInit()               settings, siteaccess, session, request/preinput and request/
input events
         headers                     defaults plus security headers
         dispatchLoop()              URL alias translation, module and view lookup, policy checks
, eZModule::run()
         pagelayout                  eZTemplate with $module_result, unless the view switched it
off
         response/output filter      the form token is added to POST forms here
         shutdown()
  -> echo $result->getContent()
```

index.php

`index.php` (the web front controller) does little on its own:

1. Sets a default time zone if `php.ini` has none, `ignore_user_abort( true )` and `error_reporting( E_ALL )`.
2. Answers maintenance mode from `var/maintenance.json` before any settings or database are read (`expMaintenance::check()`).
3. Requires `autoload.php`.
4. Runs a request filter file when `config.php` defines `EXP_REQUEST_PREPEND_FILE`. It runs on every request, also in a persistent worker, where `config.php` itself runs only once.
5. Shows PHP errors in the browser when `eZINI::environment()` is `dev` or `test` (from `EXP_ENV` in `config.env.php`).
6. Runs the kernel: `$kernel = new ezpKernel( new ezpKernelWeb() ); echo $kernel->run()->getContent();`

autoload.php

Every entry point starts here, the web and the command line alike:

Step	Why it matters to you
defines `EXP_ROOT_DIR`	the installation root as an absolute path; use it instead of paths computed from `__DIR__` of a kernel file, which break when the kernel runs from the engine archive (The engine archive)
loads `lib/phpcompat.php` when `PHP_VERSION_ID < 80100`	functions newer than PHP 8.0 exist on 8.0 too (Writing code that runs on PHP 8.0)
reads `EXP_ENGINE_PHAR`	whether kernel and library classes come from `dist/engine.phar`
unregisters the `phar://` stream wrapper	unless the runtime itself is read through it; an image upload can never be included as a phar
registers the autoloader	`autoload/ezp_kernel.php`, `var/autoload/ezp_extension.php` and, with `EZP_AUTOLOAD_ALLOW_KERNEL_OVERRIDE`, `var/autoload/ezp_override.php` (Autoloads and class maps)

ezpKernelWeb: initialization

`ezpKernelWeb` (`kernel/private/classes/ezpkernelweb.php`) is constructed with the request and prepares it:

- reads `site.ini` and finds the **siteaccess** (by host, port or URI), which decides the settings in effect;
- starts the session (`sessionInit()`) and the request (`requestInit()`), which notifies `request/preinput` and then `request/input` (`ezpEvent::getInstance()->notify( 'request/input', ... )`); the form token check of `ezformtoken` is a listener of `request/input`;
- sends the default headers (`Cache-Control`, `Content-Type`, `Content-language`, the security headers of `ezpKernelWeb::securityHeaders()`), overridable per path by `eZHTTPHeader`.

A POST without a valid form token ends here with a 403 page, before any module runs (CSRF protection and ezformtoken).

The dispatch loop

`ezpKernelWeb::dispatchLoop()` turns the address into a module result:

1. **URL translation.** When the URL translator is enabled, `eZURLAliasML::translate()` turns `/about-us` into `content/view/full/<node>`, then wildcards (`eZURLWildcard::translate()`) are tried.
2. **Module lookup.** `eZModule::accessAllowed()` checks the siteaccess's module and view lists, `eZModule::exists()` loads the module's `module.php`.
3. **Access.** The user needs `user/login` for the siteaccess, then `hasAccessToView()` checks the view's `functions` against the user's policies. Views in `[RoleSettings] PolicyOmitList[]` skip this.
4. **The view.** `$module->run( $functionName, $params )` runs the view file through `eZProcess::runFile()`, which gives it `$Params` (with `$Params['Module']`) and takes back `$Result`. A class-based view returns the same array (Module views as classes).
5. **Redirects and errors.** A view can redirect (`$Module->redirectTo()`) or return an error (`$Module->handleError( eZError::KERNEL_NOT_FOUND, 'kernel' )`), which loops once more through the error module.

The pagelayout and output

Unless the view switched it off (`$Result['pagelayout'] = false`; a template name there picks another pagelayout), the kernel creates `eZTemplate::factory()`, sets `$module_result` (your `$Result` with `content`, `path` and more) and renders the siteaccess's `pagelayout.tpl` (View results and the pagelayout). The output then passes the `response/output` filter, where `ezformtoken` adds the token to every `method="post"` form, and becomes an `ezpKernelResult`.

Shutdown

`shutdown()` finishes the request's line in the request logs (`expRequestLog::finish()`), runs the clean-up handlers registered with `eZExecution::addCleanupHandler()` (`eZExecution::cleanup()`), marks the exit as clean and stores the expiry timestamps (`eZExpiryHandler::shutdown()`). A view that sends its own output (a download, JSON) ends early with `eZExecution::cleanExit()`; under Velocity that call ends the request by an exception the server handles, so never wrap it in `catch ( Exception $e )` (Velocity persistent workers for PHP developers).

Events on the way

Event	Kind	When
`request/preinput`	notify	before the request input is processed
`request/input`	notify	the request is known; input checks such as the form token
`module/functionlist`	filter	the list of a module's views and functions

Event	Kind	When
`content/view`	filter	the node id of a content view, before it is loaded
`response/preoutput`, `response/output`	filter	the rendered page before it is sent; return the value

How to listen: Events, listeners and filters.

The command line is different

Command line scripts do not run `ezpKernelWeb`. They start with `autoload.php` and then `eZScript`: `startup()`, `getOptions()`, `initialize()` (siteaccess, extensions, session, database) and `shutdown()`. `[Event] Listeners[]` are attached to web requests only; `[RunnableSettings] Listeners[]` reach commands too (Write a console command).

Related pages

- A tour of the source tree
- Front controller pattern
- Kernel architecture
- 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.