Extension development: the book (start here)

Last changed 2026-11-04
/exphelp/manual/help-gen-manual-extension-development

A complete book on growing from user to builder, extension developer and theme developer with Exponential: finding ideas, building extensions with and without RAD tools, upgrading, forking ethically, releasing.

This book teaches you, step by step, to change what Exponential can do. It starts where you are, as a user of the administration, and takes you through four steps: **builder** (settings, templates, overrides), **extension developer** (modules, operators, datatypes, workflow events, cronjobs and more), **theme developer** (design extensions), and **maintainer** (releasing your work and contributing to other people's). Along the way it shows you where the community's existing extensions are, so that you start from what others built rather than from nothing, and how to reuse their work fairly.

Every command and code sample in the book was checked against the kernel it describes: the classes, methods and settings it names exist, the PHP samples pass `php -l`, and the template samples were rendered by the template engine. When a page states a fact, it names the file or the page where the fact comes from.

Official Exponential certifications are planned, part of the project's future plans for the community, partners, Exponential developers and students; they are not available yet (Certifications and certificates).

Developing on the whole stack: Velocity is part of Exponential 6.0.15

As of 6.0.15 the Velocity Internet / Web Application Server is part of Exponential: the root `composer.json` requires `se7enxweb/exponential-velocity ~0.0.4.57` (through `se7enxweb/exponential-php-floor`, on PHP 8.0 and later), so your development installation has the same server production runs, on PHP 8.0 and later. For an extension developer that changes three things:

- **You develop against the engine that serves the site.** Velocity's own server is the recommended engine for every stage, from your laptop to production (FrankenPHP is the production-ready alternative). Start it with `./console exp:velocity start` after `[ServerSettings] Engine=qbix`.
- **Workers keep your code loaded.** A changed class or setting reaches the site after `./console exp:velocity deploy` (or `restart`), which regenerates the autoloads, clears the caches and restarts the workers in the right order; `./console exp:velocity cache clear --apply` empties the response cache. Write code that keeps no per-request state in statics or globals, so it behaves the same under persistent workers and under PHP-FPM.
- **More is in the box.** The RAD tools write a working extension in minutes, and Velocity adds WebSockets and Server-Sent Events, a scheduler and a response cache without another package. Exponential is one working stack (server, RAD platform, CMS, digital experience, e-commerce), not just a framework you complete yourself.

The story: Velocity is part of Exponential 6.0.15. The engines: Velocity engines. The RAD tools: RAD tools.

The parts

Part	For	What you learn

Part	For	What you learn
Part 1: From user to developer	everyone	What an extension is, the tools, a development installation, reading the code and the logs, finding existing extensions on GitHub, Packagist, Share Exponential, Exponential Projects and eZpedia, and reusing them within their licenses
Part 2: The builder	builders, designers, application creators	Settings and their scopes, templates, the design chain, overrides, view modes, fetch, content classes, roles, workflows, cronjobs, all without PHP
Part 3: The extension developer	developers	The anatomy of an extension, chapter by chapter: metadata, settings, autoloads, modules and views, policies, operators, fetch functions, datatypes, workflow events, cronjobs, commands, designs, ajax, translations, tables, events, handlers, security, CSP, admin pages, tests, documentation
Tutorials	developers	Real small extensions built step by step, with the RAD tools and by hand
Part 4: Upgrading an old extension	developers	Bringing an extension written for older PHP and older releases to Exponential 6, with before and after examples from extensions this project modernized
Part 5: Extending, forking and contributing	developers, designers	Extending without forking, forking and rebranding ethically, licenses, attribution, pull requests, adopting abandoned extensions, the maintainer path
Part 6: Theme developers	designers	Themes as design extensions, with the RAD wizard and by hand; assets, the packer, CSP, dark mode, distribution
Part 7: Releasing your extension	maintainers	Version numbers, metadata, Composer and Packagist, changelogs, GitHub releases, packages, the release checklist
Part 8: Reference	developers	The kernel classes, APIs, operators, functions, settings and commands an extension author uses most

Where to start

You are	Start with
An editor or administrator who wants to build	Part 1, then Part 2
A PHP developer new to Exponential	What an extension is, then Part 3 and the tutorials
A designer	Part 2, then Part 6
An application creator	Part 2, then the capstone tutorial
Someone with an old extension to bring up to date	Part 4
About to publish an extension	Part 7 and Part 5

Reading paths through the book gives complete reading orders for each role, and the book index lists every topic from A to Z.

Fast: what can I build, and how quickly?

You want	Fastest way	Chapter
A new value or function for templates	a template operator, with the RAD wizard in minutes	With RAD: a template operator in five minutes
A page with its own address	a module with a view	By hand: a module with a view

You want	Fastest way	Chapter
A new kind of field for content	a datatype	With RAD: a datatype
Something to happen when content is published	a workflow event	With RAD: a workflow event
A job that runs every night	a cronjob part	By hand: a cronjob part
A new look	a design extension	Part 6
The site's settings under version control	a settings-only extension	With RAD: a settings-only extension

The RAD tools live at **Setup > RAD** (`/setup/rad`); they list every place the kernel can be extended and write a working extension for most of them. Every tutorial also shows the same extension written by hand, so you understand what the wizard wrote and can work without it.

Conventions in this book

- Commands run from the installation root. `./console` is a shortcut to `bin/php/console` (how to create it). Add `--allow-root-user` when you run a command as root.
- Example extensions are called `mysite_hello` and similar, and new classes start with `exp` or your own prefix, never with `ez`: that prefix belongs to the kernel's existing classes.
- Administration paths are written as `Setup > RAD` and addresses as `/setup/rad`.
- The product is **Exponential**. It continues the code base of eZ Publish, which its creator discontinued after almost 18 years; kernel identifiers such as `eZINI` and file names such as `ezinfo.php` keep their historical names.

Demo mode. Code that sets or changes user passwords should respect demo mode (`EXP_DEMO_MODE=1` on public demos): Demo mode for developers.

Related pages

- Develop with Git: grow from user to developer
- Extensions: find, install, configure, build and release
- RAD tools
- Contribute your extensions back
- Who is 7x?, Who is the Exponential Foundation?
- Free documentation: this book 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. Start with Partners: start here and How to apply.

Reading paths through the book

Reading orders through the book for each role - curious user, builder, PHP developer, designer, application creator, upgrader, maintainer - with what you can do after each stage.

This page is part of Extension development: the book. The book is long, and nobody needs all of it at once.

Pick the path that matches what you want to do. Each path lists the chapters in order and says what you can do when you reach the end of each stage.

The curious user: "What can be changed, and by whom?"

1. The ladder from user to developer
2. What an extension is
3. Building a site without code
4. Finding ideas on GitHub and projects.exponential.earth

After this you can tell a setting from a template from an extension, and find extensions others wrote.

The builder: settings, templates and overrides, no PHP

1. Tools you need and setting up a development installation
2. How INI settings work, then Settings scopes and their order and Siteaccess settings
3. Templates for builders, The design chain, Template overrides, View modes
4. Fetch for builders, Template operators builders use most
5. Content classes, Sections, roles and access, Workflows, Cronjobs
6. When settings are not enough

After this you can shape a site's behavior and look without writing PHP, and keep your work in an extension.

The PHP developer new to Exponential

1. What an extension is, Reading the code you have, Debug output and logs
2. Part 2, chapters 1 to 6 of the builder path (settings, templates, the design chain)
3. Part 3, the anatomy, in order
4. The RAD tools tour, then the tutorials pairwise: each "With RAD" tutorial and its "By hand" twin
5. Capstone: a complete extension
6. Security in extensions and Content Security Policy for extensions
7. Keep Part 8 open while you work

After this you can write and test an extension of any kind the kernel supports.

The designer

1. The builder path, chapters 1 to 3
2. Part 6, in order
3. admin4 styling for extension pages if you style admin pages
4. Where the designers' help continues

After this you can build, ship and maintain a theme.

The application creator

1. The builder path
2. Modules and views, Persistent objects and your own tables, Policy functions and access
3. Ajax with ezjscore server functions and expservices for apps that talk to the site
4. Content functions from PHP for imports
5. The capstone tutorial

The upgrader: "I have an old extension"

1. Evaluating an old extension and Reading licenses before you reuse anything
2. Part 4, from the checklist on, in order
3. The four case studies at the end of Part 4
4. When to fork, and when not to if the extension is someone else's

The maintainer

1. Part 5: forks, credit, licenses, pull requests, reviews
2. Part 7: versions, tags, Packagist, changelogs, the release checklist
3. The maintainer path and The maintenance life cycle

Related pages

- Previous: Extension development: the book
- Next: Part 1: From user to developer
- Book index: every topic from A to Z
- 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. Start with Partners: start here and How to apply.

Part 1: From user to developer

The first part of the extension development book.

Part 1: From user to developer

The first part of the extension development book.

This is the first part of Extension development: the book. It is written for people who already use Exponential, as an editor, an administrator or a site owner, and who want to start changing what it can do. You do not need to know PHP yet. By the end of this part you will have a development installation of your own, you will know how to read the code that is already on your disk, and you will know where to look for the work of the

hundreds of people who built extensions before you, and how to reuse it fairly.

What this part covers

Chapter	What you learn
What an extension is	The folder, the switch that turns it on, and the dozen kinds of things it can add
The ladder: user, builder, extension developer, theme developer	The four steps of the ladder and which part of this book belongs to each
The tools you need	A terminal, Git, PHP, Composer, an editor, a browser with developer tools
Setting up a development installation	One command to a working site that you can break without consequences
Your first hour as a developer	A guided hour: switch on debug output, read a template, change a setting, undo it
Reading the code you already have	How the installation is laid out, and how Git tells you why a line is there
Finding the extensions installed on your site	Setup > Extensions, the About page and the command line
The se7enxweb extension catalogue	The 66 extension packages the kernel requires, what each does, where its source is
Finding ideas on GitHub	Searching code, reading history, judging whether a project is alive
Finding ideas on Packagist	Composer packages of the extension type, versions and dependencies
Share Exponential	The community portal: forums, tutorials, books, channels
Exponential Projects	The project forge with thousands of historical extensions
eZpedia	The free community encyclopedia, its solutions and snippets
Learning from the free books	Three books about the code base, two of them under the GNU FDL
Reading licenses before you reuse anything	GPL, MIT, GFDL, "all rights reserved" and "no license"
Linking versus copying	When to quote, when to link, how to give credit
Evaluating an old extension	A checklist before you install or adopt something written years ago
Learning from an extension, step by step	Two small real extensions taken apart, file by file
Debug output and logs for developers	Seeing what the system does: debug output, template lists, log files
Asking the community for help	Where to ask, how to ask so people can answer

How to read it

Read the chapters in order the first time. Each one ends with **Previous** and **Next** links. Every command was run on an Exponential 6 installation before it was written down, and every page names the file or the page where its facts come from, so you can check them yourself.

> **Tip**
>
> Keep a terminal and the administration open side by side while you read. Nearly every chapter asks you to look at something on your own installation; seeing it there is what makes it stick.

After this part

- If you want to change a site without writing PHP, continue with Part 2: The builder.
- If you already change settings and templates comfortably, jump to Part 3: The extension developer.
- If you are a designer, Part 6: Theme developers is your shortcut.

Related pages

- Extension development: the book
- Next: What an extension is
- Develop with Git: grow from user to developer
- Extensions: find, install, configure, build and release
- 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. Start with Partners: start here and How to apply.

What an extension is

A folder under extension/ that adds settings, templates, modules, datatypes, workflow events and more without touching the kernel, and does nothing until switched on.

Part 1, chapter 1. This chapter is for anyone who has heard the word "extension" and wants to know exactly what it means in Exponential: what it is on disk, how it is switched on, and what kinds of things it can add. Everything you build later in this book is an extension, so this is the foundation.

In one sentence

An extension is **a folder under** `extension/` that adds settings, templates, modules, content types, commands, cronjobs and many other things to Exponential **without changing the kernel**, and that does nothing until it is **switched on**.

The folder

Look at your own installation:

Shell

```
ls extension/
```

You see one folder per extension, for example `ezjscore`, `ezoe`, `eztags`, `explayouts`, `sevenx_themes_simple`. The folder name is the extension's name everywhere: in settings, in Composer's `extra` block, in the extension list of the administration. The setting that names the folder is `ExtensionDirectory` in the `[ExtensionSettings]` block of `settings/site.ini`, and its default value is `extension`.

Inside, an extension uses the same directory names the kernel uses, so once you know one you can read all of them:

Folder or file	What it holds
`ezinfo.php`, `extension.xml`	Who made it, which version, which license (see ezinfo.php)
`settings/`	`.ini.append.php` files that add to or change the kernel's settings
`classes/`	PHP classes, found through the autoload array
`autoloads/`	Template operator registrations (`eztemplateautoload.php`)
`modules/<name>/`	A module with views: pages with their own address
`design/<design>/templates/`	Templates, and `stylesheets/`, `javascript/`, `images/`
`datatypes/<name>/`	A datatype: a new kind of field for content classes
`eventtypes/event/<name>/`	A workflow event type
`cronjobs/`	Cronjob scripts
`bin/php/`	Command line scripts
`translations/<locale>/`	Translations of the extension's texts
`share/`, `sql/`	Database schema and data, packages

The complete map is in The extension directory layout.

The switch

An extension on disk does nothing. It works once it is listed in `ActiveExtensions[]` of `[ExtensionSettings]` (for the whole installation) or in `ActiveAccessExtensions[]` (for one siteaccess, that is one site or the administration). See which ones are switched on:

Shell

```
./console exp:ini list site.ini/ExtensionSettings --allow-root-user
```

You see one `ActiveExtensions[]=<name>` line per active extension, in loading order. The order matters: an extension loaded earlier wins over a later one for settings and design templates. The browser way is **Setup > Extensions** (`/setup/extensions`); both are explained in Extensions: find, install, configure, build and release.

Why the kernel stays untouched

The kernel is the code under `kernel/`, `lib/`, `settings/` and `design/` at the root of the installation. It is replaced at every upgrade. Anything you change there is lost, or worse, merged badly into the next version. An extension lives beside the kernel and plugs into it through places the kernel was built to look:

- a **setting** that names your folder or your class (for example `ExtensionRepositories[]` for modules, `DesignExtensions[]` for templates, `ExtensionDirectories[]` for datatypes),
- a **directory** the kernel scans for a known file name,
- a **design** in the design chain, so your template is found before the kernel's,
- an **event** the kernel announces, that your listener answers.

Because of this, an upgrade of the kernel leaves your extension alone, and an extension can be shared: copy the folder to another installation, switch it on, and it works there too.

What an extension can add

The kernel knows more than seventy places where an extension can plug in. **Setup > RAD** (`/setup/rad`) lists every one of them in seven groups, drawn from the catalogue in `kernel/setup/expradcatalogue.php`. When this page was written that catalogue held 73 entries:

Group	Entries	Examples
Content	9	datatype, content class, rich text custom tag, information collection behavior
Templates and design	8	template operator, fetch function, template function, design extension, override set
Modules and views	11	module, view of an existing module, policy function, REST provider, ajax function
Workflow, events and jobs	9	workflow event type, trigger, notification handler, cronjob part, event listener
Storage and infrastructure	12	database handler, cluster file handler, search engine, session handler, mail transport
Packaging and shop	9	package handler, payment gateway, VAT and shipping handlers
Users, access and language	15	login handler, content policy limitation, translation, icon theme, siteaccess settings

Count the entries on your own installation at `/setup/rad`; the number grows when the kernel learns a new place. The curated description of every entry is Extension points.

Small or large, both are extensions

- **Very small**: `extension/str_replace` is two PHP files and one settings file. It adds one template operator.
- **Medium**: `extension/eztags` adds a datatype, modules, templates, fetch functions and its own database tables.
- **Very large**: `extension/explayouts` and its sibling packages add a visual page builder with an editor, an API and its own storage.
- **Settings only**: an extension can hold nothing but `.ini.append.php` files, for example the settings of one site. That is a perfectly good extension and the safest way to keep a site's configuration in version control.
- **Design only**: a theme is an extension with a `design/` folder (see Themes are extensions).

What an extension is not

It is not	Because
A change to a file in `kernel/`	That is lost at the next upgrade. Use a setting, an override or an event instead.
A Composer library	A library under `vendor/` is plain PHP code. An extension can *be* installed by Composer (its package type is `ezpublish-legacy-extension`), but it still lands in `extension/`.
Content	Content lives in the database. An extension can *create* content classes or ship sample content in a package, but the content itself belongs to the site.
Automatically safe	An extension runs with the same rights as the kernel. Read Evaluating an old extension before you install one you do not know.

Try it

1. List your extensions: `ls extension/ | wc -l` counts the folders on disk.
2. Count the active ones: `./console exp:ini get site.ini/ExtensionSettings/ActiveExtensions --allow-root-user`.
3. Pick one active extension and list its folders: `ls extension/<name>/`. Match each folder to the table above.

The difference between the two counts is normal: installations carry extensions they do not use, so that switching one on is a settings change rather than an installation.

Related pages

- Previous: Part 1: From user to developer
- Next: The ladder: user, builder, extension developer, theme developer
- Extensions: find, install, configure, build and release
- The Extensions page
- RAD tools
- 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. Start with Partners: start here and How to apply.

The ladder: user, builder, extension developer, theme developer

Four steps from using Exponential to extending it - user, builder, extension developer, theme developer - what each one does, what each needs to know, and which part of this book teaches it.

Part 1, chapter 2. People grow into development with Exponential in small, safe steps. This chapter names the steps, says what you do and need to know on each, and points to the part of the book that teaches it. Find yourself on the ladder, then read upward.

The four steps

Step	You change	You need	Book part
User	Content: articles, images, menus, users	the administration	Part 1 (this part)
Builder	Settings, templates, overrides, content classes, roles, workflows	a text editor, the INI and template languages, patience	Part 2
Extension developer	New modules, datatypes, operators, workflow events, cronjobs, commands	PHP, the kernel's classes, Git	Part 3, Part 4
Theme developer	The look: a design extension with templates, CSS, JavaScript	HTML, CSS, the template language, the design chain	Part 6

Two more roles sit beside the ladder rather than on it:

- **Maintainer**: you publish your extension so others install it, and you look after it. Releasing is Part 7; contributing to other people's work and growing the community is Part 5.

- **Application creator**: you combine content classes, extensions and settings into a complete application (a members' area, a shop, a booking system). You use all four steps, mostly the builder's and the extension developer's.

Step 1: user

You already know the administration. What helps you climb:

- Know **where things are configured**. Each page of the administration is drawn from templates and governed by settings. When you wonder "why does it do that?", the answer is almost always a setting or a template.
- Know **what is installed**. Finding the extensions installed on your site shows you how.
- Learn to **read a log**. When something goes wrong, the error log names the cause far more often than people expect. See Debug output and logs for developers.

What you can already do without code: create content classes, sections, roles and policies, workflows, URL aliases, translations. Each of those is a builder's tool too.

Step 2: builder

A builder changes how the site behaves and looks **without writing PHP**:

- **Settings** (`.ini` files): which extensions run, which design a site uses, how images are scaled, which languages exist, what is cached.
- **Templates** (`.tpl` files): the HTML of every page, written in the template language with variables, loops, conditions, operators and fetch functions.
- **Overrides**: a rule that says "for articles in this section, use *this* template instead of the standard one".

The builder works in an extension too: a settings extension and a design extension keep every change in one folder under version control, instead of scattered through `settings/override/`. Part 2 teaches all of it.

> **Note**
>
> Most sites need nothing beyond the builder's tools. Do not write PHP until a setting, a template or an override really cannot do the job. When settings are not enough helps you decide.

Step 3: extension developer

You write PHP that the kernel calls:

- a **module** with views (a page with its own address, for example `/report/list`),
- a **template operator** or **fetch function** that templates can call,
- a **datatype** (a new kind of field),
- a **workflow event** (something that happens when content is published),
- a **cronjob part**, a **command**, an **ajax function**, a **handler** that replaces a part of the system.

You learn the anatomy of an extension and the kernel's classes (`eZINI`, `eZContentObject`, `eZTemplate` and the rest, see Part 8). The RAD tools in **Setup > RAD** write a working skeleton of most of these for you, so you start from code that runs; the tutorials in Part 3 show both the RAD way and the hand way.

Step 4: theme developer

You build the look of a site as a **design extension**: templates, stylesheets, scripts, images, and the override rules that use them. You work with the design chain, the stylesheet packer, dark mode tokens and the Content Security Policy that forbids inline scripts. Part 6 covers what is special about themes; the Designers section of the help goes much deeper into theme design and templates.

How long does each step take?

That depends on you, and this book makes no promises about hours. What it does promise: each step builds only on the one below it, each chapter ends with something you can check on your own installation, and nothing asks you to learn something you do not need yet.

A sensible order

1. Read Part 1 fully, and do Your first hour as a developer.
2. Build something real with Part 2: a settings extension for your own site.
3. Do the first three tutorials in Part 3: the smallest extension by hand, a template operator and a module with a view.
4. Take an old extension that you like and bring it up to date with Part 4. Nothing teaches the kernel faster.
5. Publish it (Part 7) and offer your changes to its author (Part 5).

Reading paths through the book gives ready-made orders for developers, designers and application creators.

Related pages

- Previous: What an extension is
- Next: The tools you need
- Develop with Git: grow from user to developer
- Reading paths through the book
- 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. Start with Partners: start here and How to apply.

The tools you need

The short list of tools for extension development with Exponential - a terminal, PHP 8, Composer 2, Git, a text editor, a browser with developer tools, xmllint and PHPUnit - with the command that checks each one.

Part 1, chapter 3. Extension development needs very little. This chapter lists the tools, says what each is for, and gives the one command that tells you whether you have it. Nothing here costs money.

The list

Tool	What for	Check	You need at least
A terminal	Every command in this book	open one	any shell (bash, zsh)

Tool	What for	Check	You need at least
PHP (command line)	Running the console, the installers, `php -l`	`php -v`	8.0 (the kernel's `composer.json` accepts 8.0 and later 8.x versions; 8.5 is supported)
Composer	Installing Exponential and extensions as packages	`composer --version`	2.x
Git	History, branches, sharing your work	`git --version`	2.28 (see Develop with Git)
A text editor	Writing PHP, templates, settings		anything that saves plain UTF-8 without a byte order mark
A browser with developer tools	Looking at the HTML, CSS, network requests and console errors your pages produce	press F12	any current browser
`xmllint`	Checking `extension.xml` before a release	`xmllint --version`	part of libxml2
PHPUnit	Running tests (the kernel's and your own)	installed by Composer as a development requirement	the kernel's `composer.json` accepts 9.6 to 13.x

The requirements of Exponential itself (PHP extensions, databases, web servers) are in Requirements. This chapter is only about what *you* need on top.

PHP on the command line

You run PHP far more often on the command line than you think: the console (`./console`), the scripts under `bin/php/`, and the syntax check you will use after every change:

Shell

```
php -l extension/mysite_hello/ezinfo.php
```

Expected: `No syntax errors detected in extension/mysite_hello/ezinfo.php`. Make this check a habit; it costs a second and catches the mistake that would otherwise take a whole page down.

> **Note**
>
> The command line PHP and the PHP that serves your web pages can be different programs with different settings. When something works on the command line but not in the browser (or the other way round), compare the two: **Setup > System information** shows the web server's PHP.

Composer

Exponential and its extensions are Composer packages. You need Composer to install them and to install the development tools (PHPUnit). Set it up once so that every `se7enxweb/*` package comes as a Git clone you can read and change:

Shell

```
composer config --global 'preferred-install.se7enxweb/*' source
```

Why and how is explained in Develop with Git. Composer's own documentation is at getcomposer.org.

Git

Git keeps the history of every file. For a developer it answers three questions you will ask every day: *what did I change?* (`git diff`), *why is this line here?* (`git log -p`, `git blame`), and *how do I go back?* (`git restore`, `git revert`). Every extension in the se7enxweb organization is a Git repository with its full history. Learn the basics at git-scm.com.

A text editor

Use whatever you like. What matters:

- **UTF-8 without a byte order mark.** A byte order mark before `<?php` is output before the page starts and breaks headers. `./console exp:ezcheckphptag` finds files with characters before the PHP start tag.
- **Line endings** `LF`, not `CRLF`.
- **Syntax highlighting** for PHP, and if your editor knows it, for Smarty-like templates; the template language is close enough to Smarty that a Smarty mode helps.
- **Search across files.** You will search the kernel a lot. On the command line, `grep -rn` does the same:

Shell

```
grep -rn "function fetchByNodeID" kernel/classes/
```

A browser with developer tools

The administration and your site are web pages. The developer tools (F12) show:

- **Elements**: the HTML your templates produced. Turn on the template path comments (see Debug output and logs for developers) and the HTML tells you which template drew each part.
- **Console**: JavaScript errors and Content Security Policy violations. Exponential's administration enforces a Content Security Policy, so an inline script in your template shows up here as a violation, not as a working button.
- **Network**: which stylesheets and scripts were loaded, and the response of an ajax call.

Tools that are already in the installation

You do not need to install these; they come with Exponential:

Tool	Command	Use
The console	`./console list`	Every command of the kernel and of active extensions, with a description
Settings tool	`./console exp:ini`	Read and change settings, see which file wins
Autoload generator	`php bin/php/ezpgenerateautoloads.php -e`	Make the system find your new classes
Cache tool	`php bin/php/ezcache.php --clear-all --allow-root-user`	Clear caches after a change
Class check	`php bin/php/checkclasses.php --allow-root-user`	Load every class and list those PHP refuses
Code style	`./console exp:checkcodestyle`	Check changed PHP files against `phpcs.xml.dist`

Tool	Command	Use
Run with bootstrap	`php bin/php/ezexec.php <script.php>`	Run your own script with the kernel loaded
RAD tools	`Setup > RAD` (`/setup/rad`)	Wizards that write extensions, catalogue, health check

`--allow-root-user` is only needed when you run a command as the root user; as a normal user, leave it out.

Optional but useful

- **A database client** for the database your installation uses, to look at tables while you learn. Read, do not write: the content tables are maintained by the kernel and break when edited by hand.
- **PHPStan and PHP_CodeSniffer**: the kernel's own quality checks use them (`phpstan.neon.dist`, `phpcs.xml.dist` in the root). See Tests for extensions.
- **A GitHub account**, once you want to publish or contribute. Installing needs none.

Check everything at once

Shell

```
php -v && composer --version && git --version && xmllint --version 2>&1 | head -1
```

Expected: four version lines. If one is missing, install it with your system's package manager before you continue.

Related pages

- Previous: The ladder: user, builder, extension developer, theme developer
- Next: Setting up a development installation
- Requirements
- Develop with Git
- 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. Start with Partners: start here and How to apply.

Setting up a development installation

A development installation in a few commands: Git clones, Composer, exp:install on SQLite, Velocity, debug settings, and the rules that keep it apart from live.

Part 1, chapter 4. You never develop on a live site. This chapter sets up an installation of your own where you can break things, read the debug output and try every example in this book. It takes a few commands; everything after the clone is the same whether you work on a laptop, a virtual machine or a container.

What you get

- The kernel and every extension as **Git clones**, so you can read their history and keep your changes on a branch.
- A database in **one SQLite file**, so there is no database server to set up.
- A **web server** started with one command.
- **Debug output** switched on, so errors are shown instead of hidden.

Step 1: get the code as Git clones

Tell Composer once to install every `se7enxweb/*` package as a Git clone:

Shell

```
composer config --global 'preferred-install.se7enxweb/*' source
```

Then clone the kernel, start your own branch and install the packages:

Shell

```
git clone https://github.com/se7enxweb/exponential.git
cd exponential
git switch -c my-dev
composer install
```

The first `composer install` takes a few minutes: it clones every extension the kernel's `composer.json` requires into `extension/<name>`, each a Git repository of its own. The full explanation, including how to update later, is in Getting the code and Develop with Git.

Check:

Shell

```
ls -d extension/*/.git | wc -l
git -C extension/eztags log --oneline -3
```

Expected: a number of extension clones, and the last three commits of one of them.

Step 2: install with one command

Shell

```
php bin/php/console exp:install --apply
```

With no options this installs the `sevenx_multisite` site package on **SQLite**, in `eng-US`, and prints a summary with the site address, the administration login and a generated administrator password (also written to `var/log/initial-admin-password`, readable by its owner only). Every option, and the other databases, are documented in The console install.

Many developers add a shortcut so that `./console` works from the root:

Shell

```
ln -s bin/php/console console
./console list | head
```

The shortcut is not part of the distributed files; this book writes `./console` and assumes you made it.

> **Warning**
>
> `exp:install` refuses to run over an existing installation unless you pass `--force`, and its default `--db-action=remove` empties the database it is given. On your own development machine with SQLite that is what you want. Never point it at a database that holds anything you care about.

Step 3: start a web server

The quickest is Velocity with PHP's built-in server engine, the shipped default:

Shell

```
./console exp:velocity start
```

`settings/velocity.ini` ships with `[ServerSettings] Engine=php`, which runs `php -S` with the router `bin/php/velocity-router.php`; its port is set in `[PHPServerSettings]` (8087 by default, see Serving the site). Open the address the summary of `exp:install` printed, then the administration. For an Apache or nginx setup, the same chapter has complete examples.

Step 4: make mistakes visible

A live site hides its errors from visitors. A development site should show them. Make the changes in **your own settings extension or the override directory of your development machine**, never in `settings/site.ini` itself:

Shell

```
./console exp:ini set site.ini/DebugSettings/DebugOutput enabled global --allow-root-user
./console exp:ini set site.ini/TemplateSettings/ShowUsedTemplates enabled global --allow-root-use
r
./console exp:ini set site.ini/TemplateSettings/TemplateCompile disabled global --allow-root-user
php bin/php/ezcache.php --clear-tag=ini --allow-root-user
```

Setting	Default	On a development machine	What it does
`[DebugSettings] DebugOutput`	`disabled`	`enabled`	Errors, warnings, timing and SQL appear below each page
`[TemplateSettings] ShowUsedTemplates`	`disabled`	`enabled`	The debug output lists every template a page used, and which override chose it
`[TemplateSettings] TemplateCompile`	`enabled`	`disabled` while you edit templates	Templates are read fresh on every request instead of from the compiled cache
`[TemplateSettings] ShowTemplatePathComments`	`disabled`	`enabled` when you hunt for a template	HTML comments in the page source name the template of each part

`exp:ini set ... global` writes `settings/override/site.ini.append.php` and keeps a backup; `--dry-run` shows the change first. The settings are explained in Debug output and logs for developers.

> **Tip**
>
> Debug output is shown to everyone who can open the page unless you limit it. On a machine only you can reach, that is fine. Anywhere else, use `DebugByIP=enabled` with `DebugIPList[]`, or `DebugByUser=enabled` with `DebugUserIDList[]`, both in `[DebugSettings]`.

Step 5: a place for your own extensions

Create a folder for the extension you will build in the tutorials and put it under version control from the start:

Shell

```
mkdir -p extension/mysite_hello
git -C extension/mysite_hello init
```

If you prefer to keep your own extensions outside `extension/`, the kernel can search additional folders; see Extensions in more than one folder.

Rules that keep development and live apart

Rule	Why
Never develop on the live site	A typo in a template or a settings file takes the page down for every visitor.
Never copy `settings/override/` from live to development or back	It holds database passwords and host-specific values. Each machine keeps its own.
Use a copy of live content only when you must	Live content can contain personal data. A fresh `exp:install` with the demo package is usually enough.
Commit your extension, not the kernel	Your work lives in `extension/<yours>`; the kernel is updated through Composer and Git, not by your edits.
Turn debug output off before anything goes live	Debug output reveals paths, SQL and settings.

When something goes wrong

Symptom	What to check
`composer install` stops with a PHP version error	`php -v`; the kernel needs PHP 8.0 or later (PHP 7.4 usable without support)
`exp:install` stops before the summary	its own message names the check that failed; the log is in `var/log/`
The site shows a blank page	`var/log/error.log`, and the PHP error log of your web server
A change to a template has no effect	template compilation: clear the template cache, or set `TemplateCompile=disabled` while developing
A new class is not found	`php bin/php/ezpgenerateautoloads.php -e`, then clear the caches

Related pages

- Previous: The tools you need
- Next: Your first hour as a developer
- Getting the code, The console install, Serving the site
- Develop with Git
- 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. Start with Partners: start here and How to apply.

Your first hour as a developer

A guided hour: see who sets a setting, change and undo it, find the template behind a page, read the debug output and a log, and run PHP with the kernel loaded.

Part 1, chapter 5. This chapter is a guided hour on the development installation from the previous chapter. You do six small exercises, each one with something to type and something to see. Nothing here writes code yet; the point is to learn how to *look* at the system, which is most of a developer's work.

> **Note**
>
> Do this on your development installation only. Every exercise can be undone, and the last section shows how.

Exercise 1: where does a setting come from? (10 minutes)

Every behavior of Exponential comes from settings in `.ini` files, and many files can set the same value. The question "which file wins?" is the first one a developer learns to ask.

Shell

```
./console exp:ini where site.ini/SiteSettings/SiteName --allow-root-user
```

You see every file that sets `SiteName`, in the order they are read, and the value in effect at the end. The kernel's default in `settings/site.ini` comes first; overrides, siteaccess settings and extensions come later and win.

Now the same for a setting an extension changes:

Shell

```
./console exp:ini where design.ini/ExtensionSettings/DesignExtensions --allow-root-user
```

That is the list of extensions whose `design/` folders the template system searches. Each line names the extension that added itself. This is how extensions plug in: by adding a line to a setting.

Exercise 2: change a setting, and undo it (10 minutes)

Shell

```
./console exp:ini set site.ini/SiteSettings/SiteName "My dev site" global --dry-run --allow-root-
user
```

`--dry-run` prints the change as a diff and writes nothing. Run it again without `--dry-run`. The command writes `settings/override/site.ini.append.php`, keeps a copy of the old file under `var/backup/ini/<timestamp>/`, and clears the settings cache. Reload a page of your site: the name changed.

Undo it:

Shell

```
./console exp:ini rem site.ini/SiteSettings/SiteName global --allow-root-user
./console exp:ini get site.ini/SiteSettings/SiteName --allow-root-user
```

The value in effect is the one from before. You just did the most common builder's task, safely and reversibly.

Exercise 3: which template drew this page? (10 minutes)

Switch on the list of used templates (if you did not in the previous chapter):

Shell

```
./console exp:ini set site.ini/TemplateSettings/ShowUsedTemplates enabled global --allow-root-use
r
./console exp:ini set site.ini/DebugSettings/DebugOutput enabled global --allow-root-user
```

Open the front page of your site and scroll to the bottom. The debug output now has a table of every template that was used, with the template that was *requested* and the file that was *used*. When the two differ, an override replaced the standard template. Find the pagelayout (the template that draws the frame around every page) in the list.

Then switch on the path comments, reload, and view the page source (Ctrl+U):

Shell

```
./console exp:ini set site.ini/TemplateSettings/ShowTemplatePathComments enabled global --allow-r
oot-user
```

Every part of the HTML is now wrapped in comments that name its template file. When you want to change something on a page, this is how you find the file to change. More in Template path comments.

Exercise 4: read the debug output (10 minutes)

Look at the rest of the debug output under the page:

- **Errors and warnings**, each with a label that names the part of the system that raised it.
- **Timing points**: how long each phase of the request took.
- **Template usage** from exercise 3.
- **SQL**, when `[DatabaseSettings]` `SQLOutput` is enabled (try it, then switch it off again: it is long).

Open a page that does not exist, for example `/content/view/full/999999999`. The debug output tells you why you see an error page. This habit, reading what the system says before guessing, saves more time than any other.

Exercise 5: read a log file (10 minutes)

Errors are also written to log files under `var/log/` (and `var/<site>/log/` for some installations):

Shell

```
ls var/log/
tail -n 20 var/log/error.log
```

The administration has a viewer for the same files: **Setup > Logs**, explained in The log viewer. Every error line carries a reference you can search for; see Finding an error by its reference.

Exercise 6: run a script with the kernel loaded (10 minutes)

Create a file in your own extension folder, `extension/mysite_hello/scripts/hello.php`:

PHP

```php
<?php
// Runs with the kernel loaded: php bin/php/ezexec.php extension/mysite_hello/scripts/hello.php
$ini = eZINI::instance( 'site.ini' );
$cli = eZCLI::instance();
$cli->output( 'Site name: ' . $ini->variable( 'SiteSettings', 'SiteName' ) );
$cli->output( 'Active extensions: ' . count( $ini->variable( 'ExtensionSettings', 'ActiveExtensio
ns' ) ) );
```

Check it and run it:

Shell

```
php -l extension/mysite_hello/scripts/hello.php
php bin/php/ezexec.php extension/mysite_hello/scripts/hello.php
```

`bin/php/ezexec.php` loads the kernel (settings, database, autoloads) and then runs your file. You just used two of the kernel's classes: `eZINI`, which reads settings, and `eZCLI`, which writes to the terminal. Both are described in Part 8: eZINI and eZScript and eZCLI.

Undo everything

Shell

```
./console exp:ini rem site.ini/TemplateSettings/ShowTemplatePathComments global --allow-root-user
```

Keep `DebugOutput` and `ShowUsedTemplates` on while you work through the book; switch them off with `exp:ini rem` in the same way when you are done. Every write made a backup under `var/backup/ini/`, so even a mistake is one copy away.

What you learned

You can now	With
Find which file sets a value	`exp:ini where`
Change and undo a setting safely	`exp:ini set`, `--dry-run`, `exp:ini rem`
Find the template behind any part of a page	used templates, template path comments
Read why a page failed	debug output, `var/log/error.log`, Setup > Logs
Run your own PHP with the kernel loaded	`bin/php/ezexec.php`

Related pages

- Previous: Setting up a development installation
- Next: Reading the code you already have
- Changing settings with exp:ini
- Debugging 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. Start with Partners: start here and How to apply.

Reading the code you already have

A map of an installation and four reading techniques: grep for a name, follow an address to its view, a page to its template, and ask Git why a line is there.

Part 1, chapter 6. The best teacher of Exponential is the code on your disk: more than twenty years of a working content management system, with its full history. This chapter gives you a map of the installation and four reading techniques you will use every day.

The map

From the installation root (the directory that holds `index.php`):

Folder	What is in it	Do you change it?
`kernel/`	The core: one folder per kernel module (`content/`, `user/`, `setup/`, `exphelp/` ...), and `kernel/classes/` with the content model (`eZContentObject`, `eZContentClass`, datatypes, workflow types)	Never. Read it.
`lib/`	The libraries the kernel is built on: `ezdb/` (database), `eztemplate/` (template engine), `ezutils/` (settings, debug, HTTP, system), `ezfile/`, `ezi18n/`, `ezimage/`, `ezlocale/` ...	Never. Read it.
`settings/`	The default `.ini` files. `settings/override/` and `settings/siteaccess/<name>/` hold this installation's own settings	Only through overrides, siteaccess settings or extensions
`design/`	The kernel's designs: `standard`, `base`, `admin`, `admin3`, `admin4`, `admin4l`, `editor` ...	Never. Override in your own design extension
`extension/`	Every extension, each in its own folder	Your own extensions, yes
`var/`	What the installation writes: caches, compiled templates, logs, uploaded files (`var/<site>/storage`)	Never by hand, except to read logs
`bin/php/`	Command line scripts; `./console` runs them by name	Never. Your own go in `extension/<yours>/bin/php/`
`share/`	Translations, database schema and data, icons, the file manifest `filelist.md5`	Never
`autoload/`	The generated class maps (`ezp_kernel.php`, and the extension map)	Never; regenerate them
`doc/`	The documentation, including this book	Improve it with a pull request
`tests/`	The kernel's test suites	Read them; they are executable examples
`vendor/`	Composer libraries (Zeta Components and others)	Never

Two rules follow from the table: **you read the kernel, you do not edit it**, and **your work lives in `extension/`**. Everything in this book respects them.

Technique 1: grep for the name

When you see a name in a template, a setting or an error message, search for it:

Shell

```
grep -rn "fetchByNodeID" kernel/classes/ezcontentobjecttreenode.php
grep -rn "\[SiteAccessSettings\]" settings/site.ini
grep -rln "ShowUsedTemplates" kernel/ lib/
```

`-r` searches folders, `-n` prints line numbers, `-l` prints only file names. For a class, the autoload map tells you the file directly:

Shell

```
grep -n "'eZContentObjectTreeNode'" autoload/ezp_kernel.php
```

Technique 2: from an address to the code

Every page of the administration is a **module** and a **view**: `/content/view/full/2` is the module `content`, the view `view`, with the parameters `full` and `2`. To find the code:

1. The module's definition is `kernel/<module>/module.php` (for kernel modules) or `extension/<ext>/modules/<module>/module.php` (for extension modules).
2. In it, `$ViewList['<view>']` names the script (`'script' => 'view.php'`) and the parameters.
3. The script is next to `module.php`.

Try it:

Shell

```
grep -n "ViewList\['view'\]" -A6 kernel/content/module.php
```

You see the view's script and its parameters. The same three steps work for every module in the system; the chapter Modules and views explains what you see.

Technique 3: from a page to its template

Turn on `ShowTemplatePathComments` (see Your first hour as a developer) and view the page source. Each comment names a template file, for example `design/standard/templates/node/view/full.tpl`. Open it; that is the HTML of that part of the page. If the comment names a file inside an extension, the extension overrides the kernel's template there.

Technique 4: ask Git why a line is there

The kernel's repository has more than 27,000 commits. Each one says what changed and why. When a line puzzles you:

Shell

```
git log --oneline -5 -- kernel/classes/ezcontentobjecttreenode.php
git log -p -S "ShowTemplatePathComments" --oneline -- settings/site.ini | head -40
git blame -L 100,120 kernel/content/view.php
```

- `git log -- <file>` lists the commits that touched a file.
- `git log -S "<text>"` finds the commit that added or removed a piece of text.
- `git blame` names the commit behind each line; `git show <sha>` shows the whole change.

Every extension installed as a clone has its own history:

Shell

```
git -C extension/eztags log --oneline -10
```

Commit messages in this project start with `Added:`, `Updated:`, `Fixed:`, `Removed:` or `Renamed:`, which makes them easy to scan: `git log --oneline | grep '^[0-9a-f]* Fixed:'` lists bug fixes.

Reading the tests

`tests/tests/` holds the kernel's PHPUnit tests, organized like the kernel. A test is an executable example: it shows how a class is meant to be called and what it returns. When the documentation of a class is thin, read its test:

Shell

```
ls tests/tests/kernel/classes/ | head
grep -rln "eZINI::instance" tests/tests/lib | head -5
```

Reading the documentation

The `doc/` folder is organized by kind:

Folder	What it holds
`doc/eng-us/help/`	Pages like this one, for the help center in the administration
`doc/eng-us/guides/`	Task guides for administrators and developers
`doc/eng-us/features/6.0/`	One page per feature of the 6.0 line, with its settings
`doc/eng-us/bc/6.0/`	Behavior changes and upgrade notes, written for developers
`doc/eng-us/specifications/6.0/`	Specifications of larger subsystems
`doc/eng-us/install/`	The installation manual
`doc/changelogs/`	What changed in each release
`doc/eng-us/history/`	The chronicle of the project

The help center searches all of them; type a class or setting name into its search.

Exercise

Pick one page of the administration you use often. Find its module and view (technique 2), its template (technique 3), and the last commit that changed its script (technique 4). Write the three file paths down; you have just read your first piece of the kernel.

Related pages

- Previous: Your first hour as a developer
- Next: Finding the extensions installed on your site
- Develop with Git
- The extension directory layout
- 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. Start with Partners: start here and How to apply.

Finding the extensions installed on your site

Setup > Extensions, Info > About and the command line: what is installed, and which extension is behind a page, a template, a setting or an operator.

Part 1, chapter 7. Before you look elsewhere for ideas, look at your own installation: it probably already carries dozens of extensions, some of them switched off. This chapter shows three ways to list them, and how to find out which extension is behind a page, a template or a setting.

Way 1: Setup > Extensions

Open **Setup > Extensions** (`/setup/extensions`). The page has one list: the active extensions first, in loading order and numbered, then the inactive ones by name. Each card shows the name, version, description, license and website (from `extension.xml`, else `ezinfo.php`), whether the folder is a Git checkout, the declared dependencies (**Requires**, **Uses**, **Extends**) and **Needed by**. **Details** adds the author, the designs siteaccesses use, the last change and downloads.

The filter above the list (**All**, **Active**, **Inactive**, **Problems**) and the sort are kept in the address, so `/setup/extensions/(show)/active` is a bookmark of the active ones. Everything on the page is explained in The Extensions page.

> **Tip**
>
> **Inactive** is where ideas hide. An extension that is on disk but switched off was usually installed for a reason; read its description before you build the same thing again.

Way 2: Info > About

Info > About (`/ezinfo/about`) lists the extensions active in the current siteaccess, in the order the kernel loads them, with the version and license each states. The counts on the dashboard, on About and on the update dashboard all come from one function, `eZExtension::extensionCounts()`, and each page says what it counts (active here, active anywhere, on disk).

Way 3: the command line

Shell

```
ls extension/
./console exp:ini get site.ini/ExtensionSettings/ActiveExtensions --allow-root-user
./console exp:ini scopes extension --allow-root-user
```

The last command lists one scope per extension and marks the inactive ones, for example `extension:autonotifications` `extension` `extension/autonotifications/settings` `(inactive` `extension)`.

Which extension is behind it?

A page (module)

The address `/<module>/<view>` names a module. Extensions declare where their modules are with `ExtensionRepositories[]` in `[ModuleSettings]` of `module.ini`:

Shell

```
./console exp:ini where module.ini/ModuleSettings/ExtensionRepositories --allow-root-user
```

Each line names the extension that added a module repository. Then `ls` `extension/<ext>/modules/`

shows its modules.

A template

Turn on `ShowTemplatePathComments` (see Your first hour as a developer); a path under `extension/<ext>/design/...` names the extension. The list of extensions whose designs are searched:

Shell

```
./console exp:ini where design.ini/ExtensionSettings/DesignExtensions --allow-root-user
```

A setting

Shell

```
./console exp:ini where <file>/<Block>/<Variable> --allow-root-user
```

lists every file that sets it; files under `extension/<ext>/settings/` name the extension.

A template operator

Extensions register template operators through `ExtensionAutoloadPath[]` in `[TemplateSettings]` of `site.ini`, which points the template system at their `autoloads/eztemplateautoload.php`:

Shell

```
./console exp:ini where site.ini/TemplateSettings/ExtensionAutoloadPath --allow-root-user
grep -l "'my_operator'" extension/*/autoloads/eztemplateautoload.php
```

A datatype or a workflow event

Shell

```
./console exp:ini where content.ini/DataTypeSettings/ExtensionDirectories --allow-root-user
./console exp:ini where workflow.ini/EventSettings/ExtensionDirectories --allow-root-user
```

Read an extension's own files

Once you know the extension, its folder answers the rest:

File	Tells you
`extension.xml`	Name, version, license, website, dependencies
`ezinfo.php`	The same for older extensions, plus copyright lines that credit earlier authors
`README.md`, `doc/`	What it does, how to install it, its settings
`composer.json`	The package name on Packagist, PHP requirement, license, authors
`LICENSE`, `LICENSE.md`, `doc/LICENSE`	The license text you must respect when you reuse the code
`settings/*.ini.append.php`	Every setting it adds or changes, usually with comments
`git -C extension/<ext> log --oneline`	Its history, if it is a clone

Many extensions shipped with the distribution also have a page in the documentation: the extension catalogue and what a new installation comes with.

From templates

A template can read the same counts the dashboard shows:

```
{def $counts = fetch( 'setup', 'extension_counts', hash( 'all_siteaccesses', true() ) )}
```

Without `all_siteaccesses` the kinds that read every siteaccess's settings are left out, which keeps it fast; see The Extensions page.

Exercise

1. Open `/setup/extensions/(show)/inactive` and pick one inactive extension.
2. Read its `README.md` or `extension.xml`. What does it add: a module, a datatype, operators, a design?
3. Find one setting it ships: `ls extension/<ext>/settings/`.

You now know one more building block that is already on your disk.

Related pages

- Previous: Reading the code you already have
- Next: The se7enxweb extension catalogue
- The Extensions page
- Extension list: sort, inspect and download
- 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. Start with Partners: start here and How to apply.

The se7enxweb extension catalogue

The 65 extensions the kernel's composer.json requires, grouped by purpose, with repository, license and documentation: the first place to look for ideas and examples.

Part 1, chapter 8. The kernel of Exponential does not come alone: its `composer.json` requires **66 packages of the se7enxweb organization**, 65 extensions and the Composer installer that puts extensions into `extension/`. It suggests a dozen more. This chapter lists them by what they do. For a developer they are two things at once: building blocks you can use, and working, maintained examples of every kind of extension this book teaches.

The list was taken from the kernel's `composer.json` on the day this page was written. To see the current one:

Shell

```
grep -o '"se7enxweb/[^"]*"' composer.json | sort -u
```

Repositories are on GitHub under github.com/se7enxweb, packages on Packagist under packagist.org/packages/se7enxweb. The license column is what GitHub detects in each repository; where it says GPL-2.0, read the extension's own `LICENSE` and `ezinfo.php` for "or any later version". Pages in the documentation exist for many of them; the column "Doc" links them.

Page building: Exponential Layouts

A visual page builder ported to Exponential from the upstream layouts packages. Thirteen packages that show how a large feature is split into a domain core, an API, a user interface and pluggable query handlers.

Package	What it does	License	Doc
explayouts	Layouts, zones, blocks and rules: storage, resolver, renderer	GPL-2.0	
explayouts_core	Core domain layer (layout, rule, collection, block and zone services)	GPL-2.0	
explayouts_api	PHP API adapters between content and Layouts	GPL-2.0	
explayouts_standard	Standard block handlers (title, text, button, image ...)	GPL-2.0	
explayouts_ui	Administration user interface for Layouts	GPL-2.0	page
explayouts_ui_api	JSON API and single page editor shell	GPL-2.0	page
explayouts_site_api	Bridge between Layouts and the site API	GPL-2.0	
explayouts_relation_list_query	Relation list collection query handler	GPL-2.0	
explayouts_tags_query	Tags collection query handler	GPL-2.0	
explayouts_content_browser	Content browser back end	GPL-2.0	
explayouts_content_browser_core	Content browser core services	GPL-2.0	
explayouts_content_browser_ui	Server-rendered content browser user interface	GPL-2.0	
expquery_translator	Search query parser and node filter, ported from a query translator library	GPL-2.0	

Site framework

Package	What it does	License	Doc
expsite_api	Simple PHP services over the content model for site code	GPL-2.0	
expsite_core	Site helpers: breadcrumb path, redirect, mail	GPL-2.0	
expsite_app	A holder for one project's design overrides and customizations	GPL-2.0	
expsite_data_media	Site data for the media site	GPL-2.0	
expsite_installer	One-command site bootstrap	GPL-2.0	

Administration and editing

Package	What it does	License	Doc
exp_adminui	An alternative administration interface (AdminUI)	GPL-2.0	feature
expui	The JavaScript API of the admin: jQuery 4 and the Exp namespace, replacing YUI	GPL-2.0	
ezautosave	Saves drafts while editors type	GPL-2.0-or-later	page
ezie	Image editor inside the editing interface	GPL-2.0-or-later	page
ezmultiupload	Upload many files at once by drag and drop	GPL-2.0-or-later	page
ezwt	The website toolbar: edit from the public site	GPL-2.0-or-later	page
expchangeclass	Change the content class of an object or a whole class	GPL-2.0	
ezownerchange	Reassign the owner of content	GPL-2.0	
powercontent	Content creation tools for templates and code	GPL-2.0	
sevenx_dse	Database source editor in the administration	GPL-2.0	page
git_manager	Git checkouts, branches and backups from the browser	GPL-2.0	page
ezupdate	Package manager: Composer, Packagist and package servers in the admin	GPL-2.0	page

New kinds of fields (datatypes)

Package	What it does	License	Doc
eztags	Taxonomy: tags as a datatype, with modules and fetch functions	GPL-2.0	page
birthday	A date stored as YYYY-MM-DD	GPL-2.0-or-later	page
enhancedezbinaryfile	An extended file field	GPL-2.0-or-later	page
enhancedselection2	An extended selection field	GPL-2.0-only	page
exp_enhanced_link	A link field to content or URLs	GPL-2.0-or-later	page
ngclasslist	A field that stores a list of content classes	GPL-2.0	page
ezgmaplocation	A map location field	GPL-2.0-or-later	page
ezstarrating	Star ratings	GPL-2.0-or-later	page

Users and security

Package	What it does	License	Doc
sevenx_authentication_2fa	Two-factor sign-in and social login handlers	GPL-2.0-or-later	page
ezmbpaex	Password expiry and validation	GPL-2.0	page
hcaptcha	hCaptcha as a datatype	GPL-2.0	page
recaptcha	reCAPTCHA as a datatype	GPL-2.0	page

Search, SEO and statistics

Package	What it does	License	Doc
bcgooglesitemaps	XML sitemaps	GPL-2.0-or-later	page
bcwebsitestatistics	Page view and order statistics	GPL-2.0-or-later	page
xrowmetadata	Meta data and sitemaps for search engines	GPL-2.0-or-later	page

Import, export and integration

Package	What it does	License	Doc
bccie	Export of collected information	GPL-2.0	page
ezodf	Import and export of office documents	GPL-2.0-or-later	page
xrowextract	Content import and export (CSV, XML, JSON, packages)	GPL-2.0	page
syndication	Content synchronization between installations	GPL-2.0	page
ezprestapi	REST API v2 content provider	GPL-2.0	page
cjw_tmv	Tools for an external web service integration	GPL-2.0	
expqueue	A database-backed job queue	GPL-2.0-or-later	page

Communication, commerce, intelligence, databases

Package	What it does	License	Doc
cjw_newsletter	Newsletters	GPL-2.0	page
ezpaypal	PayPal payment gateway for the shop	GPL-2.0	page
expai	Exponential Intelligence: AI-assisted content creation	GPL-2.0-or-later	page
ezoracle	Oracle database driver	GPL-2.0-or-later	

Template tools

Package	What it does	License	Doc
owsimpleoperator	Write template operators as plain methods; expose permitted PHP functions	GPL-3.0-or-later	page
swark	Many template operators and a few workflow events	GPL-2.0-or-later	page
ezurlmultisite	URL operators for links across sites	GPL-2.0	

Designs and site packages

Package	What it does	License	Doc
sevenx_themes_simple	The default design theme "Simple"	GPL-2.0	page
sevenx_themes_media	The media site theme	GPL-2.0	
cjw_themes_jumper	A converted media-site design theme	GPL-2.0	
ezwebin	The classic website interface design	GPL-2.0-or-later	page
ezflow	Page blocks and zones, the older page builder	GPL-2.0-or-later	page

Package	What it does	License	Doc
ezdemo	The demo site design	GPL-2.0-or-later	page

The installer

Package	What it does
exponential-legacy-installer	The Composer installer for packages of type `ezpublish-legacy-extension`: it puts each one into `extension/<name>`

Suggested, not required

The kernel's `composer.json` also suggests (installs only when you ask) these extensions, among others: `ezfind` (Solr search), `ezauthorize` (payment gateway), `ezsi` (edge side includes), `ezscriptmonitor` (long running scripts), `bcurlaliaswithdash`, `sevenx_valkey` and `sevenx_valkey_cache` (Valkey and Redis), `adminaid`, `ezpm` (private messages). Read the `suggest` block of `composer.json` for the current list and one line on each.

Extensions that ship *inside* the kernel's own repository are not in either list: `ezjscore`, `ezoe`, `ezformtoken` and `expservices`. They are always there.

How to use this catalogue as a developer

You want to learn	Read
A small template operator	`str_replace` (in the kernel tree), then `owsimpleoperator`
A datatype	`birthday` (small), then `eztags` (large)
A module with views and policies	`ezownerchange`, `expchangeclass`
Workflow events	`swark` (`eventtypes/event/`)
ezjscore ajax functions	`ezstarrating`, `ezautosave`
A design extension	`sevenx_themes_simple`
A large feature split into packages	the Layouts packages
Modern JavaScript without inline scripts	`expui`

Learning from an extension, step by step takes two of them apart.

Related pages

- Previous: Finding the extensions installed on your site
- Next: Finding ideas on GitHub
- The extension catalogue in the documentation
- What a new installation comes with
- 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. Start with Partners: start here and How to apply.

Finding ideas on GitHub

Search GitHub for extensions and working code: the se7enxweb organization, code search for kernel class names, history and issues, and whether a project is alive.

Part 1, chapter 9. Almost every extension ever written for this code base that still exists is on GitHub, and the se7enxweb organization keeps a large share of them, original and forked. This chapter shows how to search there for ideas and working code, and how to judge what you find.

The se7enxweb organization

github.com/se7enxweb holds the kernel, every extension the kernel requires, the documentation sites, the free books and a large number of forks of historical extensions kept so they do not disappear. When this page was written (October 2026) the organization listed more than a thousand repositories; fewer than a hundred were original repositories of the organization, the rest forks.

Kind	How to recognize it	Use it for
Maintained extension	not archived, recent commits, a release tag, a Packagist package	install and build on it
Kept fork	marked as a fork, often archived	read it for ideas; check its license; bring it up to date before use (Part 4)
Documentation and books	names like `doc.exponential.earth`, `Book-...`	learn
Distributions and skeletons	names like `exponential-platform-...`	see how complete installations are put together

On a repository page, **Archived** means read-only: nobody will merge your fix there. A fork shows "forked from" and the original owner; the original is where the history started.

Search in the browser

GitHub's search understands qualifiers. Some useful searches (type them into the search field on github.com):

You look for	Search
Repositories of the organization about tags	`org:se7enxweb tags`
Datatypes written for this code base	`"extends eZDataType" language:PHP`
Template operators	`"eZTemplateOperatorArray" language:PHP`
Workflow event types	`"extends eZWorkflowEventType" language:PHP`
Cronjob parts	`"[CronjobPart-" path:settings`
ezjscore ajax functions	`"extends ezjscServerFunctions"`
Uses of one kernel method	`"eZContentFunctions::createAndPublishObject"`

Code search needs you to be signed in. A search for a kernel class name is the fastest way to find real code that uses it, which is often clearer than any description.

Search from the terminal

The GitHub command line tool `gh` does the same from a terminal:

Shell

```
gh search repos --owner se7enxweb tags --limit 20
gh search code "extends eZDataType" --language php --limit 20
gh repo view se7enxweb/eztags
gh api repos/se7enxweb/eztags/releases --jq '.[0].tag_name'
```

`gh repo view` prints the description and the README; the last command prints the newest release tag.

Read before you take

When you find something promising, read these before you copy a single line:

1. **The license.** The repository page shows the detected license in its sidebar; the `LICENSE` file is what counts. No license means no permission to copy. See Reading licenses before you reuse anything.
2. **The history.** `Commits` on the repository page, or after a clone:

   Shell

   ```
   git clone https://github.com/se7enxweb/<repo>.git
   git -C <repo> log --format='%ad %s' --date=short | head -20
   ```

   A long, steady history with clear messages is a good sign. A single "initial import" commit means the history was lost; the code may still be fine, but you know less about it.
3. **The issues and pull requests.** Open issues show known problems; merged pull requests show that someone reviews contributions.
4. **What version it was written for.** `composer.json` `require.php`, the README, `extension.xml`. Code written for PHP 5 needs the changes in Part 4 before it runs on Exponential 6.

Is it alive?

Sign	Alive	Needs adoption
Last commit	this year or last	many years ago
Archived	no	yes
Release tags	yes, following a pattern	none
Packagist package	yes, versions match the tags	none, or stuck on an old version
Issues	answered	unanswered for years
`composer.json` PHP requirement	8.x	5.x or missing

An extension that "needs adoption" is not useless; it is an opportunity. Adopting an abandoned extension explains how to take one over respectfully.

Fork, or just read?

For ideas, reading is enough. When you want to change the code and share the change, fork it on GitHub and work on a branch; Forking on GitHub, step by step shows how, and Contributing back with pull requests how to offer the change to its author.

Exercise

1. Search `org:se7enxweb datatype` and open three results.
2. For each, note: license, date of the last commit, archived or not, Packagist package or not.
3. Pick the one that looks most alive and read its datatype class. You will meet the same methods in Datatypes.

Related pages

- Previous: The se7enxweb extension catalogue
- Next: Finding ideas on Packagist
- Develop with Git
- GitHub Docs: searching code
- 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. Start with Partners: start here and How to apply.

Finding ideas on Packagist

Packagist lists every package of the extension type: how to find them, read versions, requirements and licenses, and install one on a development installation.

Part 1, chapter 10. Packagist is the public index Composer reads. Every extension that can be installed with `composer require` is listed there, with its versions, its requirements and its license. For someone looking for ideas it is a second catalogue, organized by package rather than by repository.

The extension package type

Composer packages say what they are with a `type`. Extensions for this code base use the type `ezpublish-legacy-extension`; the installer package (`se7enxweb/exponential-legacy-installer`) recognizes that type and puts the package into `extension/<name>` instead of `vendor/`. The name of the folder comes from `extra.ezpublish-legacy-extension-name` in the package's `composer.json`, for example:

JSON

```
{
    "name": "se7enxweb/str_replace",
    "type": "ezpublish-legacy-extension",
    "license": "GPL-2.0-or-later",
    "extra": {
        "ezpublish-legacy-extension-name": "str_replace"
    }
}
```

List every extension package

Packagist has a listing by type. In a browser: packagist.org/packages/list.json?type=ezpublish-legacy-extension. From a terminal:

Shell

```
curl -s 'https://packagist.org/packages/list.json?type=ezpublish-legacy-extension' | grep -o '"[^"]*/[^"]*"' | wc -l
curl -s 'https://packagist.org/packages/list.json?vendor=se7enxweb' | grep -o '"se7enxweb/[^"]*"'
```

When this page was written (October 2026), Packagist listed 398 packages of the extension type, 127 of them under the `se7enxweb` vendor, and 218 packages under `se7enxweb` in total (the rest are libraries, distributions and bundles). The numbers change; run the commands for today's.

Read a package

The page `https://packagist.org/packages/<vendor>/<name>` shows, for each version:

On the page	Why it matters to you
Versions and their dates	Is it maintained? Does the newest version match the newest tag on GitHub?
`require`	Which PHP and which other packages it needs
License	What you may do with the code
Source	The repository; read the history there
Installs	A rough sign of how many sites use it

The same as JSON, which is useful in scripts:

Shell

```
curl -s https://packagist.org/packages/se7enxweb/eztags.json | head -c 600
```

From an installation, Composer shows what it knows about an installed package:

Shell

```
composer show se7enxweb/eztags
composer show --all se7enxweb/eztags | head -30
```

Versions you can trust

A version on Packagist comes from a Git tag. A tag, once Packagist has read it, is never read again, so a version number means the same code forever. This is why the projects behind these packages never move a tag and publish a new version instead (see Tags and releases are permanent).

When a newer tag exists on GitHub but Packagist does not show it, the repository's webhook to Packagist is usually missing; The Packagist webhook explains how to check.

Install one on your development installation

Shell

```
composer require se7enxweb/<name>
```

Composer puts the package into `extension/<name>`. That does not switch it on: activate it as described in Extensions: find, install, configure, build and release, regenerate the autoloads and clear the caches. Some extensions need more (a database table, a content class); their README says so.

> **Warning**
>
> Install new packages on your development installation first, never directly on a live site. A package runs with the full rights of the kernel.

Packagist versus GitHub

Question	Look on
Can I install it with Composer?	Packagist
Which versions exist?	Packagist (tags it read), GitHub (all tags)
Why was it changed?	GitHub (history, issues, pull requests)
Is there an older, unpublished extension that does this?	GitHub forks, Exponential Projects

Related pages

- Previous: Finding ideas on GitHub
- Next: Share Exponential (share.exponential.earth)
- composer.json for an extension
- Composer and Packagist
- 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. Start with Partners: start here and How to apply.

Share Exponential (share.exponential.earth)

The community portal: forums for extensions and developers, tutorials, books, code standards, blogs and chat channels, and how to cite what you use there.

Part 1, chapter 11. Share Exponential is the community portal of the Exponential project. This chapter tells you what a developer finds there and how to use it well. The description is as of October 2026; the portal itself is the authority on its current sections.

What it is

The portal describes itself as the place where developers, users and contributors work together. Its footer states that its content is copyright 7x and that the project "is not associated with the original eZ Publish software or its original developer, eZ Systems or Ibexa". It runs on Exponential itself.

The sections a developer uses

Section	Address	Use it for
Forums: Extensions	share.exponential.earth/forums/extensions	Ask about a specific extension, announce yours
Forums: Developer	share.exponential.earth/forums/developer	Kernel APIs, PHP questions, "how do I hook into ..."
Forums: Setup & design	share.exponential.earth/forums/setup-design	Templates, overrides, designs

Section	Address	Use it for
Forums: Install & configuration	share.exponential.earth/forums/install-configuration	Installation and settings
Forums: Translation	share.exponential.earth/forums/translation	Translating the kernel and extensions
Learn	share.exponential.earth/learn	Tutorials, links to the books and documentation, code standards
Code standards	share.exponential.earth/learn/exponential-code-standards	How code for the project is written
Write a tutorial	share.exponential.earth/learn/write-a-tutorial	Give back what you learned
Get involved: Develop	share.exponential.earth/get-involved/develop	How to contribute code
Blogs	share.exponential.earth/blogs	News from the team and contributors
Roadmap	share.exponential.earth/roadmap	Where the project is going

The **Project** menu links the downloads, the pull requests and the repository on GitHub, the project forge (Exponential Projects) and the chat channels. The Telegram channel t.me/exponentialcms is also named in Who is 7x?; the portal lists Slack, Discord and a LinkedIn group as well.

How to search it

1. Use the portal's own search for words from your error message or the name of the class or setting.
2. Search the forums of the old product name too: many answers from the earlier years still apply to the kernel, whose code base Exponential continues.
3. When you find an answer, check which version it was written for. Settings and templates rarely changed; PHP code written for PHP 5 needs the changes in Part 4.

Using what you find

The portal's content is copyright 7x unless a page says otherwise. That means:

- **Link** to a forum thread or tutorial rather than copying it into your own documentation.
- **Quote** a short passage when you need to, with a link and the date you read it.
- **Code** posted in a forum carries no license statement by itself. For a few lines that solve a common problem this is rarely a concern, but for anything substantial, ask the author in the thread whether you may reuse it under your extension's license. Reading licenses before you reuse anything explains why.

Giving back

A portal like this lives on answers. Once you have solved something:

- Answer the question in the forum where you found it unanswered.
- Write a tutorial (the **Write a tutorial!** page explains how).
- Announce your extension in the Extensions forum when you release it; see Announcing a release.

Related pages

- Previous: Finding ideas on Packagist
- Next: Exponential Projects (projects.exponential.earth)

- Asking the community for help
- Who is the Exponential Foundation?
- 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. Start with Partners: start here and How to apply.

Exponential Projects (projects.exponential.earth)

Exponential Projects, the forge that replaced projects.ez.no: thousands of projects by type, how to read an entry, and how to reuse what you find without breaking copyright.

Part 1, chapter 12. Exponential Projects is a project forge: a directory of extensions and other projects with their news, downloads, forums and source links. It calls itself "a responsive projects.ez.no replacement portal", and much of its directory comes from that older forge, which makes it the largest single index of historical extensions for this code base. Facts on this page are as of October 2026.

What is there

When this page was written the portal counted **3,934 registered projects**. They can be browsed in several ways:

Browse	Address	Shows
By type	/types	Projects grouped by what they are
By organization	/org	Projects grouped by who publishes them
As a tree	/tree	The whole directory
Recent source changes	/changes	The newest Git commits of projects with a connected repository
News	/news	Announcements

A project's address has the form `https://projects.exponential.earth/org/<organization>/<project>`.

The types a developer looks at

The type "eZ publish (Via Projects.ez.no)" (/types/ez_publish_via_projects_ez_no) held 338 projects, in subtypes that map one to one to the chapters of this book:

Subtype	Projects	Book chapter
Module	121	Modules and views
Datatype	73	Datatypes
Template operator	71	Template operators
Command line script	61	CLI scripts and console commands
Workflow event type	38	Workflow event types
Template function	27	Template functions
Admin interface integration	26	Admin pages in the admin4 style

Subtype	Projects	Book chapter
Content classes	20	Content classes for builders
Translation	14	Translations in an extension
Extended attribute filter	11	Fetch functions
Content action handler	10	Handlers named by settings
Payment gateway	9	Handlers named by settings
Search plugin	9	Handlers named by settings
Login handler	8	Handlers named by settings
Binary file handler	7	Handlers named by settings

Smaller subtypes include single sign-on handlers, collaboration handlers, notification events and handlers, XML custom tags, database plugins, SOAP functions, shop account, VAT and shipping handlers. A second type, "3rd party (via Projects.ez.no)", held 110 more projects; the "Exponential", "Exponential Platform" and "Exponential Platform Legacy" types hold the current distributions and a design extension.

Reading a project entry

Each entry in a listing shows the project's name, its status (for example Alpha or Stable), its "unix name" (usually the extension folder name), the last update, the version and the versions it was written for, and a short description. A project page adds news, downloads, a forum, images and, where connected, the Git or GitHub source.

The "compatible versions" field is the most important one for you: most projects in the historical types were written for PHP 5 and older releases of the kernel. Treat them as **ideas and starting points**, then follow Part 4 to bring one up to date, or search GitHub for a maintained fork first (Finding ideas on GitHub).

Copyright: link, do not copy

The portal's footer reads "Copyright ©1998 - 2026 7x (except where otherwise noted). All rights reserved." The portal's own text is therefore not free to copy: **link to project pages** instead of reproducing their descriptions.

The **projects themselves** carry their own licenses, almost always the GNU GPL, stated in their downloads (a `LICENSE` file, the header of the PHP files, `ezinfo.php`). The project's license is what governs the code, not the portal's footer. Read it before you reuse anything: Reading licenses before you reuse anything.

A workflow for finding a building block

1. Decide the kind of thing you need (a datatype, an operator, a module).
2. Open its subtype on /types and skim the descriptions.
3. For a promising project, open its page: status, compatible versions, last update, download.
4. Search GitHub for the same unix name; a maintained fork may exist under se7enxweb or elsewhere.
5. Read the license, then the code. Decide: use as is, bring up to date, or only take the idea.

Related pages

- Previous: Share Exponential (share.exponential.earth)
- Next: eZpedia (ezpedia.exponential.earth)

- Evaluating an old extension before you use it
- Adopting an abandoned extension
- 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. Start with Partners: start here and How to apply.

eZpedia (ezpedia.exponential.earth)

eZpedia, the free community encyclopedia under the GNU FDL: the articles that matter to extension developers, how old they are, and how to reuse them with credit.

Part 1, chapter 13. eZpedia is the free encyclopedia of this code base, written by its community since 2006. It holds hundreds of articles (722 English articles when this page was written in October 2026) in namespaces for learning, solutions, code snippets, projects and people. This chapter picks the articles that matter to extension developers and says how to use them.

Its license

The site states: "All text is available under the terms of the GNU Free Documentation License". That is the same family of license as this help (GNU FDL 1.3, see Free documentation), so you may quote and adapt eZpedia text, as long as you keep it under the GNU FDL and **credit the article** with its title and a link. In this book we still prefer to link: the articles are worth reading whole, in their own context.

The namespaces

Namespace	Address	What is in it
Learning	/en/learning	Concepts and how-tos: extensions, templates, designs, debugging, datatypes, workflows
Solution	/en/solution	Concrete problems and their fixes, many with code
Snippet	/en/snippet	Short pieces of code and template
Project	/en/project	Notes on extensions and tools

Articles for this book's readers

Titles are quoted as the articles name themselves; they use the product name of their time.

Topic	Article	Read with
Extensions	"Extensions"	What an extension is
Building in an extension	"Learning: How to develop eZ publish solutions within an extension"	Part 3
Troubleshooting	"HOWTO Troubleshoot eZ Publish Extensions"	Debugging extensions
Design extensions	"Learning: How to create a design extension"	Part 6
Override system	"Template Override System"	Template overrides: the basics
Template debugging	"Learning: Template Debugging"	Debugging templates
Debugging	"Debugging"	Debug output and logs for developers

Topic	Article	Read with
Pagelayout	"Pagelayout.tpl"	View results and the pagelayout
Node templates	"Custom Node Templates"	Override conditions
Datatypes	"Datatype"	Datatypes
Workflows	"Workflow"	Workflow event types
Settings	"Configuration Files"	How INI settings work
Kernel overrides	"Overriding kernel classes within an extension"	Kernel class overrides: the last resort
View parameters	"How to add view parameter support to your custom module view"	View parameters
Image aliases from PHP	"Creating image alias image variations in PHP"	Part 8 reference
Rich text from PHP	"Manipulating ezxml document contents"	Creating, publishing and changing content from PHP
Relations from PHP	"Adding related object programmatic"	the same
PHP API examples	"PHP eZ Publish API source code examples"	Part 8
Edit and action handlers	"Content edit handlers", "Content action handlers"	Handlers named by settings

How old is it?

Many articles were last changed between 2006 and 2012. The concepts (settings, overrides, the design chain, modules, datatypes, workflow events) have hardly changed since. What has changed:

Then	In Exponential 6
PHP 4 and PHP 5 code (`function ClassName()` constructors, `each()`, `create_function()`)	PHP 8.0 to 8.5 (PHP 7.4 usable without support); see Part 4
`include_once( 'kernel/classes/...' )` lines at the top of files	Autoloading; regenerate with `php bin/php/ezpgenerateautoloads.php -e`
Inline `<script>` and `onclick=""` in templates	Forbidden in the administration by its Content Security Policy
YUI and older jQuery	jQuery 4 and the `Exp` namespace of expui
Clearing caches by deleting folders	`php bin/php/ezcache.php`, `./console exp:cache`

So: trust the concepts, check the code against the kernel you have (grep for every class and method it uses), and expect to modernize it.

Improve it

eZpedia is a wiki; anyone may edit. When an article helped you but its code is out of date, the kindest thing you can do is add a note with the Exponential 6 way and a link to the page of this book that explains it.

Related pages

- Previous: Exponential Projects (projects.exponential.earth)
- Next: Learning from the free books

- Linking versus copying: citing sources fairly
- 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. Start with Partners: start here and How to apply.

Learning from the free books

Three books about the code base, kept on GitHub by se7enxweb, two under the GNU FDL: what each covers, how to read them against Exponential 6, and how to reuse them.

Part 1, chapter 14. Several books were written about the code base Exponential continues, when it was still called eZ Publish. Three of them are kept in repositories of the se7enxweb organization. They teach the concepts that have not changed (content classes and objects, the node tree, templates, overrides, siteaccesses, extensions) in the long, patient form only a book allows.

The three books

Repository	What it is	License (as stated in the repository)
Book-Learning-eZPublish	"Learning eZ Publish", described as the first book published about the system. A PDF and its sources	GNU Free Documentation License 1.3 (`LICENSE.md`, `COPYRIGHT.md`, `AUTHORS.md` in the repository)
Book-Building-An-eZPublish-Site	"Building a eZ Publish Site", building a complete site step by step. Sources	GNU Free Documentation License 1.3 (`LICENSE.md`, `COPYRIGHT.md`, `AUTHORS.md`)
Book-eZPublish-Basics-GuideBook	"eZ Publish Basics (Guide)", a mini book. A PDF	The repository description says it was released under the GNU FDL, but the repository holds no license file. Link to it; do not copy from it.

The Share Exponential portal also links a books page from its **Learn** menu (Share Exponential).

Reading them against Exponential 6

The books describe older releases. Read them for concepts and keep this table at hand:

In the books	In Exponential 6	In this book
Installing from a tarball and a web wizard	`composer`, Git clones, `exp:install`, the kickstarter or the setup wizard	Setting up a development installation
Editing `settings/override/site.ini.append.php` by hand	Still possible; `./console exp:ini` does it with a backup and a diff	Changing settings with exp:ini
The `design/` folders, the override system	Unchanged in principle; more designs (admin4, admin4l)	Part 2
Extensions with `include_once()` of kernel files	Autoloading	Autoloads and class naming
PHP 4 and PHP 5 examples	PHP 8.0 to 8.5 (PHP 7.4 usable without support)	Part 4

In the books	In Exponential 6	In this book
JavaScript in templates	Script files only, under a Content Security Policy	Content Security Policy for extension authors

The chapters of the Learning book, rewritten for Exponential 6, are in Learning Exponential, and the tutorial *Building a site* is in Building a site step by step.

Reusing their text

The two books under the GNU FDL 1.3 may be quoted, translated and adapted, under the same license, with credit. The same license covers this help, so a passage of those books can become part of a help page if you:

1. keep the GNU FDL 1.3 notice (this help's notice already is one),
2. name the book and link its repository,
3. say what you changed (for example "updated for Exponential 6"),
4. keep the authors' copyright lines; `AUTHORS.md` and `COPYRIGHT.md` in each repository name them.

The details of what the GNU FDL asks are on gnu.org and in this help's Free documentation page. When in doubt, link instead; see Linking versus copying.

Bringing a book up to date

A book under a free documentation license can be improved by anyone. If you find a chapter whose example no longer works, a pull request to the book's repository with the Exponential 6 version is a welcome contribution, and a fine first pull request for a new contributor (see Contributing back with pull requests).

Related pages

- Previous: eZpedia (ezpedia.exponential.earth)
- Next: Reading licenses before you reuse anything
- Who is 7x?
- 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. Start with Partners: start here and How to apply.

Reading licenses before you reuse anything

The licenses you meet around Exponential, in plain words: GPL 2 or later, GPL 3, MIT, GNU FDL, all rights reserved and no license, and where to find them.

Part 1, chapter 15. Every idea you find in Parts 1's other chapters comes with a license, and the license decides what you may do with the code or text. This chapter explains the licenses you will meet, in plain words, and where to find them in an extension. It is a practical guide, **not legal advice**; for a real decision about a commercial project, ask someone qualified, and always read the license text itself.

Ideas are free, code and text are not

You may always learn from what you read and build the same *idea* yourself in your own words and your own code. What a license governs is **copying**: the code, the text, the images. The rest of this chapter is about copying.

Where to look in an extension

Place	What it tells you
`LICENSE`, `LICENSE.md`, `COPYING`, `doc/LICENSE`	The license text. This is what counts.
`COPYRIGHT.md`, `doc/COPYRIGHT`	Who holds the copyright, often several holders over the years
`ezinfo.php`	`'License'` and `'Copyright'` lines; older extensions credit their original authors here
`extension.xml`	`<license>`
`composer.json`	`"license"`, an SPDX identifier such as `GPL-2.0-or-later`
The header comment of each PHP file	`@copyright` and `@license` lines
The GitHub sidebar	The license GitHub *detects*; a guess from the license file, not a statement

When these disagree, the license text and the file headers count over a summary line. GitHub's `NOASSERTION` means GitHub could not recognize the license file, not that there is none: open the file and read it.

The licenses you will meet

GNU GPL version 2 or later (the kernel's license)

Exponential is free software under the GNU General Public License version 2 or, at your option, any later version (`COPYRIGHT.md` and `LICENSE.md` of the kernel). Most extensions use the same license.

- You may use, study, change and share the code, for any purpose, also commercially.
- When you **distribute** the code or a changed version, you must do it under the same license, with its source, and keep the copyright and license notices.
- Running a site on your own server is not distributing the extension to your visitors; giving the extension to a customer, or publishing it, is.

Read it at gnu.org/licenses/old-licenses/gpl-2.0.html.

GNU GPL version 2 only

Some files say "version 2" without "or later" (GitHub shows `GPL-2.0-only`, for example `enhancedselection2`). Code under "2 only" cannot be combined into a work under GPL 3, because neither license allows the other's terms. When you combine code from several extensions, check this detail.

GNU GPL version 3 or later

For example `owsimpleoperator` (its README and `ezinfo.php` say version 3 or any later version). Code under GPL 3 can be combined with code under "GPL 2 *or later*" (the combination is distributed under GPL 3), but not with "GPL 2 only".

MIT and BSD style licenses

Short, permissive licenses: you may do nearly anything, including use in non-free software, as long as you keep the copyright notice and the license text. Code under MIT may be included in a GPL extension; keep its notice in the copied files. Some upstream packages that Exponential extensions were ported from are under MIT: Packagist lists `netgen/layouts-core` and `netgen/query-translator` (the sources of `explayouts_core` and `expquery_translator`) as MIT, while `netgen/site-bundle` and `netgen/ibexa-site-api` are listed as GPL-2.0-only. Part 5 shows how such ports give credit (Case study: porting a package from another project with credit).

GNU Free Documentation License (for documentation)

This help, eZpedia and two of the free books use the GNU FDL. You may copy, adapt and translate the text, under the same license, with credit. See Free documentation.

"All rights reserved"

A site or a file that says only "All rights reserved" gives you **no** permission to copy, beyond the short quotation that copyright law itself allows. Link to it instead. The project forge's own pages are an example (Exponential Projects); the projects listed there have their own licenses.

No license at all

Code without any license statement is not free to copy, even when it is public on GitHub. Ask the author to add a license; until they do, take only the idea.

Combining licenses: a quick table

You want to put into a GPL 2 or later extension	Allowed?
Code under GPL 2 or later	Yes
Code under GPL 2 only	Yes, but the combination can then no longer move to GPL 3
Code under GPL 3 or later	Only if the whole extension is then distributed under GPL 3
Code under MIT or BSD	Yes, keep the notice
A forum post with no license	Only the idea, unless the author agrees
Text under the GNU FDL	Into documentation, yes, under the GNU FDL; not into the code

For a deeper explanation of how licenses combine, see the FSF's license list and choosealicense.com.

Your own extension's license

Choose one before your first commit and write it in every place listed above. Most extensions in this ecosystem use "GNU General Public License v2.0 (or any later version)", which keeps your work combinable with the kernel and with nearly every other extension. How to write it into `ezinfo.php` and `extension.xml` is in ezinfo.php; the deeper story is Licenses and how they combine.

Checklist before you copy

- [ ] I found the license text, not only a summary.
- [ ] It allows copying and changing.
- [ ] It is compatible with the license of my extension.
- [ ] I keep every copyright line and the license notice in the files I copy.

[ ] I say where the code came from (see Attribution and copyright notices).

Related pages

- Previous: Learning from the free books
- Next: Linking versus copying: citing sources fairly
- Licenses and how they combine
- Contribute your extensions back, Licensing FAQ
- The license of this documentation
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

Linking versus copying: citing sources fairly

*When to link to a source, when to quote it, when you may copy and adapt it, and how to give credit in code comments, README*s*, documentation and commit messages - with the patterns this project uses.*

Part 1, chapter 16. You will learn from other people's work constantly. This chapter gives you a simple rule for using it fairly, and the patterns this project uses to give credit in code, documentation and history.

The rule

1. **Link** by default. A link sends the reader to the source, in its context, kept up to date by its author.
2. **Quote** a short passage when the exact words matter, with a link and the date you read it.
3. **Copy and adapt** only when the license allows it, and then **keep the notices and say where it came from**.
4. **Never** present someone else's work as yours.

Source	Link	Quote briefly	Copy and adapt
A page on a site that says "All rights reserved"	yes	yes, short, with credit	no
eZpedia, this help, the free books under the GNU FDL	yes	yes	yes, under the GNU FDL, with credit
Code under the GPL or MIT	yes	yes	yes, keeping the license and copyright notices
A forum post without a license	yes	yes, short	only the idea, unless the author agrees
Code without any license	yes	no	no

Giving credit in code

When you copy or adapt code under a free license, keep its header and add your own line:

PHP

```
<?php
/**
 * Adapted from <project> (<link to the original file or repository>), <license>.
 *
 * @copyright Copyright (C) <years> <original holder>
 * @copyright Copyright (C) <year> <you or your organization> (changes)
 * @license GNU General Public License v2.0 (or any later version)
 */
```

Many extensions in this ecosystem carry several copyright lines in `ezinfo.php` for the same reason: each line credits someone who worked on it. `owsimpleoperator`'s `ezinfo.php`, for example, names 7x and Brookins Consulting together with OPEN WIDE, the company that wrote the extension first. Keep such lines; add yours.

Giving credit in a README or documentation

A short "Credits" or "Origin" section is enough:

```
## Origin

This extension is a port of <upstream package> (<link>), published under the <license>.
The structure of its services follows the upstream package; the code was rewritten for Exponentia
l 6.
```

The Layouts packages do exactly this: `explayouts_core`'s README says it is "an Exponential Legacy port inspired by the `netgen/layouts-core` package", and `explayouts`' README the same for `netgen/layouts-ibexa`.

Giving credit in commit messages

A commit message is permanent and public. When a commit brings in someone else's work, say so in its body:

```
Added: Initial import of the class conversion extension

Forked from <original extension> and hardened for PHP 8, with a transactional
rollback so that a failed conversion leaves the content untouched.
```

That is close to the real first commit of `se7enxweb/expchangeclass` (commit `7c3f164`), which says the extension was "forked from eZChangeclass and hardened for PHP 8". When you merge a contributor's pull request, merge it as a merge commit so their own commits, with their names, stay in the history (see Pull request etiquette).

Giving credit in a help page

This help links its sources and names files outside `doc/` as paths. When you adapt text from eZpedia or a free book into a help page:

- name the source and link it in the text or in "Related pages",
- say that you adapted it ("adapted from ... and updated for Exponential 6"),
- keep the page under the GNU FDL (every help page ends with the Free documentation line).

What fair use of an idea looks like

You read a forum post that solves your problem with a clever SQL query. You may:

- build your own solution using the same idea, written your own way,
- link the post in a comment above your code (`// Approach from <link>`),
- thank the author in the thread.

That is fair, honest and helps the next reader find the original discussion.

Related pages

- Previous: Reading licenses before you reuse anything
- Next: Evaluating an old extension before you use it
- Attribution and copyright notices
- Rebranding ethically
- 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. Start with Partners: start here and How to apply.

Evaluating an old extension before you use it

A checklist with commands to judge an extension you did not write: license, PHP 8 syntax, removed functions, access checks, escaping, SQL, inline scripts and database changes.

Part 1, chapter 17. An extension runs with the full rights of the kernel: it can read every table, every file the web server can read, and every visitor's request. Before you install one you did not write, or decide to adopt one, spend half an hour with this checklist. All commands run on a **development installation**, with the extension unpacked but not yet switched on.

1. License and origin (5 minutes)

[ ] A license file exists and allows what you plan to do (Reading licenses before you reuse anything).

[ ] `ezinfo.php` and `extension.xml` agree on name, version and license.

[ ] You know where it came from: a repository with history, a Packagist package, a project page.

2. Does it parse on PHP 8? (2 minutes)

Shell

```
find extension/<name> -name '*.php' -print0 | xargs -0 -n1 php -l | grep -v '^No syntax errors'
```

No output means every file parses. Any line printed is a file PHP 8 refuses; typical causes are listed in PHP 8 errors that stop old code.

3. Removed and deprecated functions (5 minutes)

Shell

```
grep -rnE '\b(each|create_function|split|ereg|eregi|mysql_[a-z_]+|money_format|get_magic_quotes_gpc|utf8_decode|utf8_encode)\s*\(' extension/<name> --include=*.php
grep -rnE 'function\s+([A-Za-z_]+)\s*\(' extension/<name> --include=*.php | head -50
```

The first search finds functions PHP 8 removed or deprecates; each has a replacement in Removed PHP functions and their replacements. The second lists the methods, so you can spot **old-style constructors**: a method with the same name as its class. PHP 8 no longer calls such a method as a constructor, so the object is created without being set up. The template operator extension `str_replace`, still in its original 2005 shape in the kernel tree, is exactly that case: its operator class sets its operator list in `function MyStrReplaceOperator()`, which PHP 8 never calls, and the template engine reports "Object problem with operator 'ezstr_replace'". The fix is one line, shown in Old-style constructors and static calls.

4. Can every class load? (2 minutes, after switching it on in development)

Shell

```
php bin/php/ezpgenerateautoloads.php -e
php bin/php/checkclasses.php --quiet-ok
```

`checkclasses.php` loads every class of the installation in a child process and names the ones PHP refuses. Add `--allow-root-user` when you run it as root.

5. Access control (5 minutes)

Open every `modules/<module>/module.php`:

Shell

```
grep -n "ViewList\|'functions'\|FunctionList" extension/<name>/modules/*/module.php
```

- [ ] Every view that shows or changes data names a `'functions'` entry.
- [ ] `$FunctionList` declares those functions, so they can be granted through roles.
- [ ] Views that change data only act on POST and check the form token (see ezformtoken and CSRF protection).

A view without `'functions'` can be opened by anyone who can reach the site.

6. Escaping in templates (5 minutes)

Shell

```
grep -rnE '\{\$[a-zA-Z_.]+\}' extension/<name>/design | grep -v '|wash' | head -30
```

Variables printed without `|wash` are a cross-site scripting risk when their value can come from a visitor (a view parameter, a form field, a search word). Not every hit is a bug (a number from the kernel is safe), but each one deserves a look. See Security in extensions.

7. SQL (5 minutes)

Shell

```
grep -rnE '(query|arrayQuery)\s*\(' extension/<name> --include=*.php | head -30
```

For each query built from variables:

- [ ] strings go through `$db->escapeString()`,

- [ ] numbers are cast with `(int)`,
- [ ] or better, the code uses `eZPersistentObject` instead of hand-written SQL.

8. Inline scripts and styles (2 minutes)

Shell

```
grep -rnE '<script[^>]*>[^<]|on(click|change|submit|load)=|javascript:|style="' extension/<name>/
design | head -30
```

The administration enforces a Content Security Policy: inline scripts, `on...=` handler attributes and `javascript:` links do not run there. Every hit is work for Part 4 before the extension's admin pages work.

9. What it changes in the system (5 minutes)

- [ ] `ls extension/<name>/settings/` and read each file: which settings does it change? Does it override a kernel class (`ezpkerneloverride`-style autoloads), replace a module, or add event listeners?
- [ ] `share/`, `sql/`, `update/`: does it create tables? Do they have upgrade scripts?
- [ ] `cronjobs/`: does it run anything on a schedule?
- [ ] Does it send mail, call external services, write files outside `var/`?

10. Decide

Findings	Decision
License fine, parses, access checks present, escaping fine	Use it; keep notes of what you checked
License fine, small PHP 8 or CSP problems	Bring it up to date with Part 4, then offer the fixes to its author
No access checks, unescaped output, SQL from request values	Do not use it on a live site until fixed
No license	Do not copy it; take only the idea

Write your findings down. If you adopt or fix the extension, they become the first issue in its tracker.

Related pages

- Previous: Linking versus copying: citing sources fairly
- Next: Learning from an extension, step by step
- The upgrade checklist
- Security in extensions
- 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. Start with Partners: start here and How to apply.

Learning from an extension, step by step

Two real extensions taken apart file by file: str_replace, one operator in its 2005 shape with a PHP 8 bug and its one-line fix, and owsimpleoperator.

Part 1, chapter 18. The fastest way to understand extensions is to take a small one apart. This chapter reads two real extensions from your installation, file by file, in the order the kernel reads them. You will meet the template operator mechanism here first; Template operators covers it fully later.

Extension 1: str_replace

`extension/str_replace` adds one template operator, `ezstr_replace`, that wraps PHP's `str_replace()`. Its README dates the first release to April 2005, and the code is still in that original shape, which makes it a perfect specimen: small, complete, and old enough to teach a PHP 8 lesson.

Step 1: list the files

Shell

```
find extension/str_replace -type f -not -path '*/.git/*' | sort
```

```
extension/str_replace/LICENSE
extension/str_replace/LICENSE.md
extension/str_replace/README.md
extension/str_replace/autoloads/eztemplateautoload.php
extension/str_replace/autoloads/str_replace_controloperator.php
extension/str_replace/composer.json
extension/str_replace/doc/readme.txt
extension/str_replace/ezinfo.php
extension/str_replace/settings/site.ini.append.php
```

Three files do the work: one setting, one registration, one class. The documentation sits where every extension keeps it: `README.md` on top, the rest in `doc/`. `doc/readme.txt` is plain text, as older extensions wrote it; new and updated extensions write those pages as Markdown (`.md`), in English (Documenting an extension).

Step 2: the setting that plugs it in

`settings/site.ini.append.php`:

INI

```
[TemplateSettings]
ExtensionAutoloadPath[]=str_replace
```

`ExtensionAutoloadPath[]` tells the template engine: "also read `extension/str_replace/autoloads/eztemplateautoload.php`". That is the whole plug. Verify how the engine reads it: `grep -n "ExtensionAutoloadPath" lib/eztemplate/classes/eztemplate.php`.

Step 3: the registration

`autoloads/eztemplateautoload.php`:

PHP

```php
<?php
$eZTemplateOperatorArray = array();
$eZTemplateOperatorArray[] =
  array( 'script' => 'extension/str_replace/autoloads/str_replace_controloperator.php',
         'class' => 'MyStrReplaceOperator',
         'operator_names' => array( 'ezstr_replace' ) );
```

It names the file, the class and the operator names. The engine registers the names at once but loads the class only when a template uses one of them (`eZTemplate::registerAutoloadOperators()` and `loadAndRegisterOperators()` in `lib/eztemplate/classes/eztemplate.php`).

Step 4: the class

The class has four methods the engine calls:

Method	Called	Returns
`operatorList()`	when the class is loaded	the operator names this object handles
`namedParameterPerOperator()`	when the class is loaded	`true`: the parameter list below is per operator
`namedParameterList()`	when a template calls the operator	each operator's parameters: name, type, required, default
`modify( $tpl, $operatorName, $operatorParameters, $rootNamespace, $currentNamespace, &$operatorValue, $namedParameters )`	for each call	nothing; it writes the result into `$operatorValue`

The interesting part of `modify()`:

PHP

```php
$subject = $namedParameters['subject'] !== null
    ? $namedParameters['subject']
    : $operatorValue;
$operatorValue = str_replace( $search, $replace, $subject );
```

`$operatorValue` is both the input and the output: when a template writes `{$text|ezstr_replace( 'a', 'b' )}`, the piped `$text` arrives in `$operatorValue`, and whatever the method leaves there is the result.

Step 5: the PHP 8 lesson

The class sets up its operator list in a method named like the class:

PHP

```php
function MyStrReplaceOperator()
{
    $this->Operators = array( 'ezstr_replace');
}
```

That was a constructor in PHP 4 and PHP 5. **PHP 8 no longer treats a method named like its class as a constructor**, so it is never called, `$this->Operators` stays `null`, and `operatorList()` returns `null`. Rendered on PHP 8.5 with the book's test harness, a template calling `ezstr_replace` produces a PHP warning, "foreach() argument must be of type array|object, null given", in

`eZTemplate::registerOperatorsInternal()`, followed by the template error "Object problem with operator 'ezstr_replace'", and the operator prints nothing.

The fix is one line, renaming the method to `__construct`:

PHP

```php
public function __construct()
{
    $this->Operators = array( 'ezstr_replace' );
}
```

With that change the same template prints `Hello Exponential` for `{ezstr_replace( 'world', 'Exponential', 'Hello world' )}`. This book's harness rendered both versions before this page was written. This is the most common PHP 8 problem in old extensions; Old-style constructors and static calls shows how to find and fix every case.

Step 6: the metadata

`ezinfo.php` names the extension, version `1.0.0`, the copyright of 7x and of the original author (2005), and the license. Its `info()` method is not declared `static`, which still works but differs from the convention the kernel reads best; see ezinfo.php. `composer.json` declares the package `se7enxweb/str_replace` of type `ezpublish-legacy-extension`, `GPL-2.0-or-later`, and requires `"php": ">=5.0.0"`, another sign of its age: a modernized package states the PHP versions it was tested on.

Extension 2: owsimpleoperator

`extension/owsimpleoperator` (from OPEN WIDE, GPL 3 or later according to its README and `ezinfo.php`) solves the boilerplate problem you just saw: it lets you write an operator as a plain method, and lets templates call PHP functions you permit.

Its files

Shell

```
ls extension/owsimpleoperator/autoloads extension/owsimpleoperator/settings
```

```
exampleoperator.php  eztemplateautoload.php  owezpautoloader.php  owsimpleoperator.php  phpfuncti
onoperator.php
owsimpleoperator.ini.append.php  site.ini.append.php
```

The plug is the same `ExtensionAutoloadPath[]=owsimpleoperator` in `site.ini.append.php`. The difference is in the registration: `eztemplateautoload.php` builds `$eZTemplateOperatorArray` **from settings**:

- `[PHPFunctions] PermittedFunctionList[]` in `owsimpleoperator.ini` lists PHP functions that templates may call as operators. The shipped list is `time`, `mktime`, `getdate`, `str_replace` and `str_rot13`.
- `[ClassOperators] PermittedClassOperatorList[]` lists `file;class;method` triples; each method becomes an operator.

An operator as a method

`autoloads/exampleoperator.php`:

PHP

```php
<?php
class ExampleOperator extends OWSimpleOperator
{
    public function example_sum( $number1, $number2 )
    {
        return ( $number1 + $number2 );
    }
}
```

The base class `OWSimpleOperator` implements `operatorList()`, `namedParameterList()` and `modify()` for you: it calls the method named like the operator with the template's arguments and uses the return value. To switch the example on you would add, in your own settings:

INI

```ini
[ClassOperators]
PermittedClassOperatorList[]=extension/owsimpleoperator/autoloads/exampleoperator.php;ExampleOper
ator;example_sum
```

Try the PHP function operators

On an installation where `owsimpleoperator` is active (check with `./console exp:ini get site.ini/ExtensionSettings/ActiveExtensions`), a template can write:

```
[{str_rot13( 'Exponential' )}] [{str_replace( 'world', 'there', 'Hello world' )}]
```

which renders `[Rkcbaragvny] [Hello there]`. That output was produced by the kernel's template engine when this page was written.

> **Warning**
>
> Every function in `PermittedFunctionList[]` can be called by every template with any argument. List only functions without side effects that read nothing outside their arguments. A function like `file_get_contents` would let a template read any file the web server can read; the shipped settings file says exactly that in its comment.

What you learned

Concept	Where you saw it
An extension plugs in by adding to a setting	`ExtensionAutoloadPath[]`
Operators are registered by name, loaded on first use	`eztemplateautoload.php`
The operator contract: `operatorList`, `namedParameterList`, `modify`	`MyStrReplaceOperator`
`$operatorValue` is input and output	`modify()`
PHP 8 ignores old-style constructors	`function MyStrReplaceOperator()`
A base class can remove boilerplate	`OWSimpleOperator`
Settings can widen what templates may do, so keep them narrow	`PermittedFunctionList[]`

Related pages

- Previous: Evaluating an old extension before you use it
- Next: Debug output and logs for developers
- Template operators
- By hand: a template operator
- owsimpleoperator in the extension catalogue
- 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. Start with Partners: start here and How to apply.

Debug output and logs for developers

See what the system does while it builds a page: the Exp Debug bar and its settings, used templates, path comments, the log files under var/log, and your own debug lines and logs from PHP.

Part 1, chapter 19. When an extension does not do what you expect, the system almost always says why, if you know where to look. This chapter shows the three places: the debug output under each page, the log files, and the lines you write yourself.

The debug output: the Exp Debug bar

With `[DebugSettings] DebugOutput=enabled` in `site.ini`, every page carries a report of how it was made. In Exponential 6 this report is the **Exp Debug bar**: a bar pinned to the page with a live summary of the request, and a panel with tabs for the sections of the report (messages, timing, templates, SQL and more). Its **Settings** tab shows every debug setting, the value in effect and the file it comes from, and can change it in a scope you choose; each change goes through the same engine as `exp:ini`, is logged, and can be undone from the bar. The details are in The Exp Debug bar.

The settings

All in `site.ini`:

Block	Setting	Default	Meaning
`[DebugSettings]`	`DebugOutput`	`disabled`	Show the debug report at all
`[DebugSettings]`	`DebugByIP` + `DebugIPList[]`	`disabled`	Show it only to these addresses (CIDR ranges allowed)
`[DebugSettings]`	`DebugByUser` + `DebugUserIDList[]`	`disabled`	Show it only to these user IDs
`[DebugSettings]`	`DebugRedirection`	`disabled`	Stop before a redirect so you can read the report of the page that redirected
`[DebugSettings]`	`ScriptDebugOutput`	`disabled`	Debug output for command line scripts too
`[DebugSettings]`	`AlwaysLog[]`	`error`	Which message levels are written to log files even without debug output
`[DebugSettings]`	`DeprecatedLog`	`enabled`	PHP deprecations go to `deprecated.log`, each distinct message once per request

Block	Setting	Default	Meaning
`[TemplateSettings]`	`ShowUsedTemplates`	`disabled`	List every template a page used, and which override chose it
`[TemplateSettings]`	`ShowTemplatePathComments`	`disabled`	HTML comments naming the template of each part of the page
`[TemplateSettings]`	`Debug`	`disabled`	More detail from the template engine
`[DatabaseSettings]`	`SQLOutput`	`disabled`	Every SQL query with its time

Check any of them with `./console exp:ini where site.ini/<Block>/<Setting>`.

> **Warning**
>
> Debug output reveals file paths, SQL and settings. On any machine that others can reach, restrict it with `DebugByIP` or `DebugByUser`, and switch it off before going live.

Reading the report

- **Errors and warnings** first: each has a label, usually the class or template that raised it, for example `eZTemplate @ design:node/view/full.tpl:12[5]` (file, line and column of the template).
- **Templates**: requested template, used template, and the override rule that picked it. When your override does not apply, this table says which template won instead.
- **Timing points**: where the time went.
- **SQL**: with `SQLOutput=enabled`, every query; a query that runs a hundred times is a loop that should fetch once.

The template path comments

With `ShowTemplatePathComments=enabled`, view the page source: each part of the HTML is wrapped in comments naming its template. This is the quickest way from "this part of the page" to "this file". See Template path comments.

The log files

Log files are written under `var/log/` (some installations use `var/<site>/log/` for site specific logs). The ones a developer reads most:

File	What is in it
`error.log`	Errors: always logged (`AlwaysLog[]=error`)
`warning.log`	Warnings, when logged
`notice.log`, `debug.log`, `strict.log`	The lower levels, when logged
`deprecated.log`	PHP deprecations: the to-do list for the next PHP version
`storage.log`	File storage operations
`404.log`	Pages not found (`[DebugSettings] NotFoundLog`)
`csp.log`	Content Security Policy violations: your inline script is listed here
`security.log`, `auth.log`	Security events and sign-ins

Shell

```
tail -f var/log/error.log            # follow it while you reload the page
grep -c "" var/log/deprecated.log    # how many deprecation lines
```

The administration has a viewer for the same files, with search and permalinks: **Setup > Logs**, described in The log viewer. Each error carries a reference that leads to every line of its request: Finding an error by its reference. The command line form is `./console exp:explogs`.

Writing your own debug lines

From PHP, use `eZDebug`. Each method is static and takes the message, an optional label and an optional style class (signatures from `lib/ezutils/classes/ezdebug.php`):

PHP

```
<?php
eZDebug::writeDebug( 'Value of $nodeID: ' . $nodeID, __METHOD__ );
eZDebug::writeNotice( 'Cache miss for ' . $key, 'mysite_hello' );
eZDebug::writeWarning( 'Setting MySettings/Limit is missing, using 10', __METHOD__ );
eZDebug::writeError( 'Could not load node ' . $nodeID, __METHOD__ );
```

Method	Level	Use for
`writeDebug()`	debug	temporary values while you work; remove them afterwards
`writeNotice()`	notice	information that helps reading a report
`writeWarning()`	warning	something unexpected the code recovered from
`writeError()`	error	something failed; always written to `error.log`

Use `__METHOD__` as the label: the report then names the class and method.

From a template, `{debug-log var=$node msg='node in my template'}` writes a value into the debug report (see Debugging templates).

Writing your own log file

For things you want to keep, independent of debug settings, write a log file of your own with `eZLog::write()` (`lib/ezfile/classes/ezlog.php`):

PHP

```
<?php
// static function write( $message, $logName = 'common.log', $dir = 'var/log' )
eZLog::write( 'Imported ' . $count . ' items', 'mysite_hello.log' );
```

The file appears under `var/log/`. Never write personal data, passwords or keys into a log.

A debugging routine

1. Reproduce the problem on your development installation.
2. Read the debug report: errors first, then the template table.
3. `tail` the error log while you reload.
4. Add `eZDebug::writeDebug()` lines around the code you suspect; reload; read.
5. When you understand it, remove the debug lines, fix the code, and check that the log stays quiet.

Debugging extensions goes further: class loading, autoloads, caches and the RAD health check.

Related pages

- Previous: Learning from an extension, step by step
- Next: Asking the community for help
- The Exp Debug bar
- eZDebug and logging
- When something goes wrong
- 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. Start with Partners: start here and How to apply.

Asking the community for help

Where to ask a development question (forums, chat, issue trackers, the extension's own repository), how to write a question people can answer, what never to post, and how to report a security problem privately.

Part 1, chapter 20. Sooner or later you will be stuck. The people who build Exponential and its extensions answer questions in public places, and a well asked question usually gets an answer. This chapter tells you where to ask, how to ask, and what never to post.

Before you ask

Most questions are answered by one of these, in this order:

1. The debug report and `var/log/error.log` (Debug output and logs for developers).
2. The help center's search: type the class, setting or error message.
3. The extension's own `README.md` and `doc/`.
4. `git log -S "<the text of the error>"` in the kernel or the extension: maybe it was changed recently, with a message that explains why.
5. The forums and eZpedia, searched for the error message.

If none of these answers it, ask. Say in your question what you already tried; it saves everyone a round trip.

Where to ask

Your question is about	Ask in
How to do something with the kernel's APIs	Forums: Developer
A specific extension	The extension's GitHub issues (if it is maintained), or Forums: Extensions
Templates, overrides, designs	Forums: Setup & design
A quick question, a discussion	The Telegram channel t.me/exponentialcms and the other channels listed on Share Exponential
A bug in the kernel, the administration or the bundled extensions	github.com/se7enxweb/exponential/issues

Your question is about	Ask in
A bug in the Velocity application server	github.com/se7enxweb/exponential-velocity/issues
The project, the website, the ecosystem	github.com/se7enxweb/exponential-community/issues
A security problem	**Privately**, see below

The administration has a guide for bug reports: **Help > Report an error** (`exphelp/report`) fills in a summary of the installation and opens a prepared GitHub issue. See When something goes wrong.

Before you file a bug, ask yourself whether the behavior might be deliberate: in Exponential, as in any project, some of it is open to interpretation and other sites rely on it. Changing such a negative feature needs a clear justification and several rounds of review, and an option is far easier to accept than a new default.

How to write a question people can answer

A good question has five parts:

```
What I want to do:   <one sentence>
What I did:          <the code or template, as small as possible>
What I expected:     <the result you wanted>
What happened:       <the exact error message, the debug report lines>
My environment:      Exponential <version>, PHP <version>, database <type>, design <name>
```

- **Make it small.** Cut your code down to the smallest piece that still shows the problem. Often you find the answer while doing it.
- **Paste text, not screenshots** of code and errors, in code blocks, so people can copy and search them.
- **Name versions.** `./console --version` prints Exponential's version; `php -v` the PHP version. **Setup > System information** has **Copy report as text**, which leaves out passwords.
- **Say what you tried** and what each attempt did.
- **Come back.** Answer follow-up questions, and when it is solved, say how. The next person with the same problem will find your thread.

What never to post

- Passwords, API keys, tokens, the contents of `settings/override/` or `config.php`.
- Database dumps, user lists, e-mail addresses or any personal data of your site's users or customers.
- Internal host names and IP addresses of your servers.
- Code you are not allowed to share (a client's proprietary code). Reduce it to a neutral example instead.

Check the error lines you paste: some contain paths or user names. Replace them with placeholders.

Security problems: privately

A security problem (a way to read data without permission, to run code, to take over an account) is **never** reported in a public tracker or forum. The kernel's `SECURITY.md` says how: by e-mail to `security@se7enx.com`, or through GitHub's private vulnerability reporting on the repository. You will be told whether the report is accepted and kept informed while a fix is prepared. For a third-party extension, look for its own `SECURITY.md`; if there is none, contact its maintainers privately before saying anything in public.

Give back

Once you are not stuck any more, someone else is. Answer one question a week in the forum you learned most from. Growing the community through code has more ways to help.

You finished Part 1

You know what an extension is, you have a development installation, you can read the code and the logs, and you know where to find ideas and how to reuse them fairly. Continue with Part 2: The builder.

Related pages

- Previous: Debug output and logs for developers
- Next: Part 2: The builder: settings, templates and overrides
- When something goes wrong
- Writing good issues
- 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. Start with Partners: start here and How to apply.

Part 2: The builder: settings, templates and overrides

The second part of the book.

Part 3: The extension developer

Part 3 of the book: the anatomy of an extension, one mechanism per chapter, from metadata and settings to modules, operators, datatypes, events, security and tests, followed by step-by-step tutorials.

Tutorials: build real extensions step by step

Step-by-step tutorials that build real small extensions, first with the RAD wizards of Setup > RAD, then by hand, ending with a complete extension from idea to release.

Part 4: Upgrading an old extension to Exponential 6

Part 4 of the book: bring an old extension up to Exponential 6, from the inventory and PHP 8 fixes to CSP, jQuery 4, admin4 styling, metadata and tests, with real before and after examples.

Part 5: Extending, forking ethically and contributing back

Part 5 of the book: build on other people's extensions fairly - extend without forking, fork when you must, keep credit and licenses right, and give your work back with pull requests, issues and reviews.

Writing good issues

Report a bug or ask for a feature in a way a maintainer can act on - versions, steps to reproduce, expected and actual result, logs without secrets - with a template and what to leave out.

Part 5, chapter 10. Not every contribution is code. A clear bug report is often worth more than a quick patch, because it lets the maintainer fix the cause instead of a symptom. This chapter shows what a maintainer needs from an issue, and what must never be in one.

Before you write

1. **Search** the repository's issues, open and closed, for the error message or the feature.
2. **Update.** Check that the problem is still there with the latest release of the extension and of Exponential.
3. **Make it small.** Find the shortest way to make it happen. A problem you can show in four steps gets fixed.
4. **Is it a security problem?** Then do not open a public issue. Follow the project's security policy (the kernel's is in `SECURITY.md` at the root of its repository) and report it privately.
5. **Is it on purpose?** Read what the documentation actually promises. Behavior that is deliberate or open to interpretation is a negative feature other sites may depend on: say why it does not work for you, ask for an option rather than a new default, and expect a longer discussion.

A bug report

```
Title: Change content class entry does nothing under an enforced Content Security Policy

Versions
- Exponential 6.0.15, PHP 8.3, Apache with PHP-FPM, SQLite
- expchangeclass 1.0.8
- Design: admin4

Steps
1. Set site.ini [ContentSecurityPolicy] Mode=enforce for the admin siteaccess.
2. Right-click a node in the content tree.
3. Click "Change content class".

Expected
The class selection page opens.

Actual
Nothing happens. The browser console shows:
"Refused to execute inline event handler because it violates the following Content Security Polic
y directive ..."

Notes
Works with Mode=report-only. The entry has an onclick attribute in popupmenu/classcontextmenu.tpl
.
```

Every part has a purpose:

Part	Why
A title that names the symptom and the condition	the maintainer can tell it apart from twenty others
Versions	half of all bugs depend on one
Steps from a known state	the maintainer can see it with their own eyes
Expected and actual	sometimes the "bug" is intended, and this shows the disagreement

Part	Why
Error messages as text, copied	they can be searched; a screenshot cannot
What you already found	saves time; it is fine to be unsure

A feature request

Describe the problem before the solution:

```
Title: A setting for the number of items on the overview page

Problem
Our editors manage 400 items. The overview shows 10 per page, so finding one takes many clicks.

Proposal
A setting myoldext.ini [OverviewSettings] Limit, default 10, read in modules/myoldext/overview.ph
p.

Alternatives considered
Overriding the template does not help: the limit is in the PHP view.

I can send a pull request if this is welcome.
```

"I can send a pull request" is the most welcome sentence in a feature request.

What never goes into an issue

Issues are public, often forever, and copied by search engines and archives:

- **No passwords, keys or tokens**, not even "old" ones. If you pasted one by mistake, change it at once; deleting the comment does not remove copies.
- **No personal data of your users or customers**: names, email addresses, IP addresses, content of their accounts. Replace them with made-up values (`user@example.invalid`).
- **No internal host names or addresses** of your servers.
- **No full configuration files.** Quote the few settings that matter.

Logs need the same care: cut them down to the lines about the problem, and replace anything personal.

After you open it

- Answer questions. An issue whose reporter does not answer is usually closed after a while.
- If you find a workaround or the cause, add it.
- If it turns out to be your mistake, say so and close it. That is useful too: the next person who searches finds the answer.

Related pages

- Previous: Pull request etiquette
- Next: Commit messages the project can build a changelog from
- When something goes wrong
- About issues on GitHub Docs
- 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. Start with Partners: start here and How to apply.

Growing the community through code

How code itself grows a community - publishing what you built, good first issues, documentation and translations, help pages inside your extension, and the community sites where people look for extensions.

Part 5, chapter 15. A community does not grow from announcements. It grows when people find something useful, can understand it, and can see where to help. Every extension you publish can do that. This chapter collects the habits that turn code into a place people come back to.

Publish what you build

The most effective contribution is the extension you already wrote for yourself. If it solves a problem for you, it solves it for someone else. The steps (clean it up, license, metadata, GitHub, Packagist) are in Contribute extensions back, and the release procedure in Part 7. Remove what is specific to your site (names, addresses, settings with secrets) and keep the rest.

Make it easy to start

- **A README that answers three questions** in its first screen: what it does, how to install it, how to see it working.
- **A screenshot or a short example** of the result.
- **Defaults that work**, so the first install shows something.
- **Help pages inside the extension**, which the administration's help system shows next to the kernel's (Documenting your extension and the tutorial Help pages for your extension).

Make it easy to help

- **Label issues** `good first issue` when they are small and well described. New contributors look for that label.
- **Say what help you want** in the README: translations, testing on PostgreSQL, a dark mode check.
- **Keep translations open.** Interface texts as translation strings (`i18n` with your extension's context) let anyone add a language with one file. expchangeclass gained Spanish, Hebrew, Latin and Klingon this way (commit c2a4f41, alongside German earlier).
- **Answer the first pull request of a new contributor quickly and kindly.** It decides whether there is a second.

Make it findable

Place	What it is for
Packagist	`composer require` finds it; search by keyword
GitHub topics	add topics such as `exponential` and `exponential-extension` to the repository
projects.exponential.earth	the community's project directory
share.exponential.earth	the community's sharing and discussion site

Place	What it is for
ezpedia.exponential.earth	the community's knowledge base, where articles can describe and link your extension

Part 1 describes each site and what you find there (projects.exponential.earth, ezpedia.exponential.earth).

Share what you learn

A solved problem written down saves the next person a day:

- an article on ezpedia.exponential.earth about a technique;
- a correction to this book: every page is free documentation under the GNU FDL, and improvements are welcome (Free documentation);
- an answer on share.exponential.earth to someone else's question.

Be the kind of project people want to join

- Credit everyone, every time (Attribution and copyright notices).
- Explain decisions in public, in issues and commit messages.
- Thank people.
- Never be the reason someone leaves.

Related pages

- Previous: The maintainer path
- Next: Case study: porting a package from another project with credit
- Contribute extensions back
- Who is the Exponential Foundation
- 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. Start with Partners: start here and How to apply.

Part 6: Theme developers

Part 6 of the book: themes are extensions.

Part 7: Releasing your extension

Part 7 of the book: release an extension so others can rely on it - version numbers, permanent tags, metadata bumps, Composer and Packagist, changelogs, GitHub releases, packages, tests, and a checklist.

Part 8: Reference of built-in features for extension authors

Part 8 of the book: a reference of the kernel classes, template features, settings, commands and directories extension authors use most, with signatures taken from the source and links to the full documentation.

Glossary for extension developers

The words this book uses, in one alphabetical list - from access word, append file and attribute to view mode, view parameter and workflow - each with a short definition and the chapter that explains it.

Part 8, reference 34, the last page of the book. The terms of Exponential development, as this book uses them.

Term	Meaning	See
access word	the answer of `hasAccessTo()`: `yes`, `no` or `limited`	eZUser and access checks
admin4	the current administration design, with shared components and design tokens	admin4 components and tokens
append file	an extension's or siteaccess's `<name>.ini.append.php` that adds to a kernel INI file	INI files an extension author meets
attribute	one field of a content object, of a datatype defined by its class attribute	eZContentObject
autoload array	the generated map from class names to files (`var/autoload/`, `autoload/`)	Autoloads and class naming
cache block	a part of a template whose output is cached (`cache-block`)	Cache tags, cache blocks and clearing caches
class (content class)	the type of a content object: its identifier and attributes	eZContentClass and its attributes
cluster file handler	the layer through which files under `var/` are read and written, so several servers share them	eZFile, eZDir and cluster file handlers
Content Security Policy (CSP)	a header that tells the browser which scripts and styles it may run	CSP: moving inline scripts into files
context menu	the right-click menu of the content tree, extended through `admininterface.ini`	Context menus and admin integration
cronjob part	a group of cronjob scripts run together	Cronjob parts
datatype	the code behind a kind of attribute: editing, storage, display	The eZDataType API
design	a named set of templates and files, searched in the design chain	Themes are extensions
design chain	the ordered list of designs (and their extension folders) a siteaccess searches	The design chain
design extension	an extension that contributes design folders (`DesignExtensions[]`)	The anatomy of a design extension
design token	a named value (`--a4-ink`) used instead of a fixed color or size	Dark mode and design tokens in themes
expui (Exponential UI)	the administration's JavaScript library, `Exp`	The expui JavaScript API
extension	a folder under `extension/` that adds to or changes the kernel without editing it	What an extension is

Term	Meaning	See
extension point	a registered way for an extension to plug in	Extension points at a glance
fetch function	a read operation of a module, called from templates with `fetch()`	Built-in fetch functions by module
fork	a copy of a repository that you maintain	When to fork, and when not to
form token	the per-session secret every POST of a signed-in user must carry	ezformtoken and CSRF protection
handler	a class named by a setting that provides a service (mail transport, image handler)	Handlers named by settings
kernel	the core of Exponential: `kernel/`, `lib/`, `settings/`, the shipped designs	
kernel override	an extension's replacement of a kernel class: the last resort	Kernel overrides: the last resort
module, view	a group of pages under one URL prefix, and one page of it	eZModule and module results
node	a location of a content object in the content tree	eZContentObjectTreeNode
object (content object)	one piece of content, with versions and translations	eZContentObject
override (template override)	a rule that picks another template for certain content	Template overrides
packer	ezjscore's bundling of stylesheets and scripts	Theme assets, the packer and CSP
package (`.ezpkg`)	an archive of classes, content or an extension for the package system	Packages (.ezpkg) and the package creator
pagelayout	the template that frames every page	View results, the path and the pagelayout
persistent object	a class mapped to a table row (`eZPersistentObject`)	eZPersistentObject
policy, role, limitation	a permission (module and function), a set of them given to users, and its restriction	Policy functions and access
RAD tools	the wizards of **Setup > RAD** that write extension code	The RAD tools tour
remote ID	a stable, installation-independent ID of an object or node	eZContentObject
server function	a PHP method callable through `/ezjscore/call/`	ezjscore server functions reference
siteaccess	one entry point of an installation, with its own settings and designs	Siteaccess settings
tag (Git)	a name for one commit, used as a release version; permanent once pushed	Tags and releases are permanent
template operator, template function	`{$x|operator}` and `{function}...{/function}`	Built-in template operators
trigger	the link between an operation (publish) and a workflow	Triggers and workflows for developers
Velocity	the application server of Exponential, required by `composer.json` since 6.0.15	
view mode	how content is shown: full, line, embed and your own	View modes for builders
view parameter	a `(name)/value` pair in a URL	View parameters

Term	Meaning	See
workflow, workflow event	a list of steps run by a trigger, and one step	The eZWorkflowType API

Demo mode - `EXP_DEMO_MODE=1` in the environment: administrator passwords cannot be changed or reset (the `expDemoMode` API, the store guard in `eZUser::store()`), and the installers accept the published demo password. Demo mode for developers.

Related pages

- Previous: expservices: remote services for extensions
- Next: Book index: every topic from A to Z
- Extension development: the book
- 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. Start with Partners: start here and How to apply.

GNU Free Documentation License 1.3