Extending Exponential 6.0

Source file: doc/eng-us/EXTENDING.md Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/EXTENDING.md

This guide is the map for developers who add to Exponential 6.0 without changing the kernel. It covers:

- which way to extend fits the job, and what an extension is made of;
- how to create one, by hand or with the RAD wizards, and how to switch it on;
- modules, views and policies;
- templates and the rules of the administration designs;
- datatypes, workflow events, cronjob parts and console commands;
- the APIs, the audit trail, secrets, translations and help pages;
- the website toolbar and Exponential Layouts;
- tests, and how to release.

It assumes a working installation from Installing Exponential 6.0 that is configured as in Configuring Exponential 6.0. Each section names the files and settings, shows a working example and links to the page with the full detail. This page stays the map and the deeper pages stay the reference.

Every command is run from the installation root: the directory that holds `index.php`, `autoload.php`, `bin/` and `settings/`. The command console is `bin/php/console`; many installations add the shortcut `./console`. When you run a script as `root`, add `--allow-root-user`.

> **Never change the kernel to add a feature.** Everything below lives in `extension/<name>/`, so an update of Exponential replaces the kernel and keeps your work. When a kernel class really has to change, the RAD tools offer a kernel override as the last resort (section 3), and the change belongs upstream as a pull request.

Contents

1. The quick start: a working extension in ten minutes
2. Choose the way to extend
3. Create an extension, by hand or with the RAD tools
4. Switch it on: activation, autoloads, tables
5. Modules, views and policies
6. Templates and design
7. Content, workflow and background work
8. Console commands
9. APIs, audit, secrets, translations and help
10. The website toolbar and Exponential Layouts

11. Tests and CI
12. Release it
13. Troubleshooting

1. The quick start: a working extension in ten minutes

The fastest way to a working extension is the RAD tools. You need an administrator login (the policy `setup/setup`).

1. Open **Setup > RAD** (`/setup/rad`). It lists every place Exponential can be extended, in seven groups.
2. Choose the tool of the point you want, for example **Module** (`/setup/moduleextension`).
3. Fill in the extension name (lower case letters, digits and underscores, for example `myhello`) and what to generate.
4. Press **Refresh preview**. The wizard lists every file it would write under `extension/myhello`, before it writes any.
5. Press **Create in extension/**. A wizard never overwrites an existing extension and writes nothing outside `extension/<name>`.

Then switch it on and make its classes known:

Shell

```
./console exp:ini add 'site.ini/ExtensionSettings/ActiveExtensions[]' myhello global --allow-root
-user
php bin/php/ezpgenerateautoloads.php -e
./console exp:velocity deploy --allow-root-user
```

Expected: `exp:velocity deploy` ends with every step PASS or SKIP. **Setup > Extensions** (`/setup/extensions`) now lists the new extension with its version, and the module's view answers at `/<module>/<view>`.

If you would rather see every file by hand first, the guide Extensions builds the same kind of extension file by file in section 6.1.

2. Choose the way to extend

Pick the smallest way that does the job. Each one keeps working across updates.

You want	The way	Where it lives
A different value of a setting	A settings override	`settings/override/` or `settings/siteaccess/<siteaccess>/` (see Configuring)
A different look of a page	A template override in a design	the `design/<design>/override/templates/` of an extension, with `override.ini.append.php`
A theme of its own	A design extension	`extension/<name>/design/<design>/`, listed in `DesignExtensions[]`
New pages, content types, operators, commands, jobs	A full extension	`extension/<name>/`

You want	The way	Where it lives
The same, shared with other installations	A Composer package of that extension	its own repository, type `ezpublish-legacy-extension`

An extension is a folder named after the extension. The parts it may have:

Part	What it does	Reference
`extension.xml`	Name, version, license, website and dependencies; read first by the about page	Extension metadata
`ezinfo.php`	The same facts for older readers: class `<name>Info` with a static `info()`	Extension metadata
`settings/`	`.ini.append.php` files that add to the kernel's settings, and `.ini` files of its own	Settings override placements
`design/`	Templates, stylesheets, scripts and images, one directory per design	Templates and design
`modules/`	Modules and their views	section 5
`classes/`	PHP classes, found through the autoload array	section 4
`autoloads/`	Template operators and functions	section 6
`cronjobs/`, `bin/php/`	Cronjob parts and console scripts	sections 7 and 8
`translations/`	One catalogue per language, `translations/<locale>/translation.ts`	section 9
`share/db_schema.dba`	The extension's own tables, on every database engine	section 4
`doc/`	Its pages in the help center	section 9

More than one directory of extensions is possible, for example for a vendor's own set: Additional extension directories.

3. Create an extension, by hand or with the RAD tools

By hand. Two files make an extension that the system recognizes: `ezinfo.php` and `extension.xml`, with the same version in both. The guide Extensions, section 6.1, writes a complete one with a module, a view and a template, and shows the expected result of each step.

With the RAD tools. Setup > RAD (`/setup/rad`) knows 68 extension points, 64 of them with a tool that writes a complete extension. The wizards and their addresses:

You want	Open
A design (templates, stylesheets, overrides)	`/setup/designextension`
Template operators, fetch functions, template functions	`/setup/templateoperator`
A handler that replaces a part of the system (login, mail, session, search, shop VAT and shipping, REST filters ...)	`/setup/handlerextension/<kind>`
An extension that only carries settings	`/setup/settingsextension`
A content class written as a script	`/setup/contentextension`
A datatype	`/setup/datatype`

You want	Open
A workflow event	`/setup/workflowevent`
A module with views, or a module over existing database tables	`/setup/moduleextension`, `/setup/modulewizard`
A kernel class replaced (the last resort)	`/setup/kerneloverride`

After you create one, **Setup > RAD > Extension point survey** (`/setup/radsurvey`) counts its registrations, and the health check on `/setup/rad` lists anything that is configured but cannot work, with how to fix it.

Reference: RAD tools, the list of all extension points with their settings and an example each, and RAD security.

4. Switch it on: activation, autoloads, tables

An extension does nothing until it is switched on.

- **For the whole installation:** `[ExtensionSettings] ActiveExtensions[]` in `settings/override/site.ini.append.php`.
- **For one siteaccess:** `ActiveAccessExtensions[]` in `settings/siteaccess/<siteaccess>/site.ini.append.php`.
- **In the administration: Setup > Extensions** switches extensions on and off and shows problems.

The order of the list is the order of priority: an extension loaded earlier wins over a later one for settings and templates. With `ExtensionOrdering=enabled`, the dependencies declared in `extension.xml` decide the order first (Loading order).

Shell

```
./console exp:ini add 'site.ini/ExtensionSettings/ActiveExtensions[]' myhello global --dry-run --allow-root-user
./console exp:ini add 'site.ini/ExtensionSettings/ActiveExtensions[]' myhello global --allow-root-user
```

Autoloads. Classes in `classes/` are found through an autoload array. Regenerate it whenever you add or rename a class:

Shell

```
php bin/php/ezpgenerateautoloads.php -e
```

Deploy. `./console exp:velocity deploy --allow-root-user` then runs every step a PHP change needs, in order: the autoloads, the INI cache, the template caches, a reload of PHP-FPM, a restart of Velocity and the content caches. It prints PASS, FAIL or SKIP for each step and stops at the first failure. Add `--packer` when stylesheets or scripts changed, and `--kernel` when a kernel class was added. Details: Deploying a PHP change and Deploying.

Tables. An extension that needs tables ships them in `share/db_schema.dba`, once, for every engine (MySQL and MariaDB, PostgreSQL, SQLite, Oracle and MongoDB).

- **New installation:** the setup step creates the missing tables of every active extension.
- **Existing installation:** the upgrade step `6.0.15-active-extension-tables` (`./console exp:upgrade`) does the same after you switch an extension on.

Missing tables are created; nothing is dropped or changed. Details: Extensions of a new installation.

Changes to the tables of an extension that is already installed are upgrade steps of your own: Upgrade steps for extensions.

5. Modules, views and policies

A module is a folder `modules/<module>/` with a `module.php` that lists its views. A view returns the page content. The module is found once the extension names itself in `settings/module.ini.append.php` (`[ModuleSettings] ExtensionRepositories[]=<name>`).

PHP

```php
<?php
$Module = array( 'name' => 'hello' );

$ViewList = array();
$ViewList['world'] = array( 'script' => 'world.php',
                            'functions' => array( 'read' ),
                            'params' => array( 'Name' ),
                            'unordered_params' => array( 'page' => 'Page' ) );

$FunctionList = array();
$FunctionList['read'] = array();
?>
```

- **View parameters, never the query string.** A view reads its input from its parameters: the ordered ones (`/hello/world/Ada`) and the named ones (`/hello/world/Ada/(page)/2`), in `$Params`. Never read `$_GET`, and never build links with `?key=value`. A form posts, and the view answers with a redirect (303) to the address with view parameters, so every state of a page has its own address. This is how the administration's own pages work since 6.0.15 (Search links use view parameters, View parameters).
- **Access.** A view without `'functions'` has no access control: anyone who can reach the site can open it. Declare a function in `$FunctionList`, name it on the view, and grant it in **Users > Roles and policies**. A limitation of your own for the content functions is a limitation handler (Content policy limitations of extensions). See Roles and policies and Request rules.
- **Views as classes.** New views are written as classes (`Exponential\Runnable\ModuleView`), and the plain view files keep working: Commands, cronjob parts and module views as classes.
- **A kernel module** can be replaced by a module of the same name in an extension: An extension can replace a kernel module.

The left menus. A link of your own in an administration menu goes into `settings/menu.ini.append.php` of the extension:

INI

```ini
[Leftmenu_setup]
Links[hello]=hello/world
LinkNames[hello]=Hello
PolicyList_hello[]=hello/read
```

The link is shown only to users who hold every policy of its `PolicyList_<key>[]`. Some settings of the menu sections place links:

- `LeadLinks[]` names links shown before the groups;
- `TrailLinks[]` names links that close the menu;
- `CollapseAfter=<n>` shows the first n links and puts the rest under **More**;
- `LinkIcons[]` and `LinkDescriptions[]` give a link its place in the mega menu of admin4.

Reference: The admin4 design.

6. Templates and design

- **Overrides.** A template override chooses another template for a match (class, node, section, view mode ...). It is registered in `override.ini.append.php` of the extension's siteaccess settings. Which override wins: Template override ordering.
- **Operators, fetch functions and template functions.** These are registered in `autoloads/eztemplateautoload.php` and in `settings/template.ini.append.php`. The RAD wizard `/setup/templateoperator` writes all three. Reference: Template operators and Template engines.
- **Page layout hooks.** These are places in the page layout where an extension adds markup without overriding it: Page layout hooks.
- **Icons.** An extension can bring its own icon theme: Icon themes in extensions.

The rules of the administration designs. These apply to every admin design (admin, admin3, admin4, admin4l) and to the admin pages of extensions:

- No `<style>` block and no `style=""` attribute in a template. All CSS goes in the design's `stylesheets/`, loaded through the settings, so the packer bundles it.
- Reuse the shared components first: `.a4-card`, `.a4-btn`, `.a4-table`, `.a4-toolbar`, `.a4-filterbar` and the rest.
- Colors, radius, spacing and fonts come only from the design's tokens (`--a4-*`). Every new token gets a dark value in the same change.
- Check every page in light and dark mode, at 1440 px and at 960 px with 200% zoom.

The test `tests/tests/design/AdminStylingRulesTest.php` enforces the first rules in CI. The full component and token API: admin4 styling.

7. Content, workflow and background work

You want	How	Reference
A field type of your own	A datatype: a class extending `eZDataType`, found through `content.ini.append.php` (`[DataTypeSettings] ExtensionDirectories[]=<name>`); RAD `/setup/datatype`	Datatypes
Something to happen at publish, before a sign-in, at checkout ...	A workflow event, attached to a trigger; RAD `/setup/workflowevent`	Workflows
Regular background work	A cronjob part: `cronjobs/<part>.php`, registered in `cronjob.ini.append.php` (`ExtensionDirectories[]=<name>` and `[CronjobPart-<part>]`)	Cronjobs

You want	How	Reference
Content classes shipped with the extension	A content class written as a script; RAD `/setup/contentextension`	Extension points

A cronjob part of an extension is listed by `./console list cron` as `cron:<part>` and runs with `php runcronjobs.php <part>`.

8. Console commands

Scripts of an extension are found automatically and become console commands:

```
extension/<name>/bin/php/<script>.php   → ext:<name>:<script>
extension/<name>/bin/shell/<script>.sh  → ext:<name>:sh:<script>
```

Shell

```
./console list ext:myhello
./console ext:myhello:report --help
```

New commands are written as classes (`Exponential\Runnable\Command`). Then the help, option parsing, output and exit codes are the same as in the kernel's own commands. Reference:

- Commands, cronjob parts and module views as classes;
- the contract in Runnable commands, cronjob parts and views;
- The console.

9. APIs, audit, secrets, translations and help

- **GraphQL and REST.** An extension adds to the GraphQL schema with a class implementing `expGraphQLSchemaExtension`, listed in `[GraphQLSettings] SchemaExtensionList[]` of its `settings/graphql.ini.append.php`: root queries, fields of a location and the types of its datatypes (GraphQL). The REST API is the extension ezprestapi (REST API v1 and v2). Connecting other systems through these APIs, rather than adding to them, is the subject of Integrating With Exponential 6.0.
- **Audit trail.** An extension that records events of its own registers a branch of the taxonomy:
  - the branch: `audit.ini.append.php` `[AuditEventSettings] Branches[<name>]=<class>`, a class implementing `expAuditTaxonomyBranch`;
  - optionally a source of its own (`[AuditSourceSettings] Sources[]`).

  Reference: The audit trail and the audit guide.
- **Secrets.** Never store a password or key in a setting. Keep the name of a secret and ask the secret store for its value: `expSecrets::get( $name )`, or `expSecrets::required( $name, $what )` when the tool cannot work without it. An administrator writes the value in **Setup > Secrets**. Reference: The secret store.
- **Translations.**
  - Every text of a template goes through `|i18n( '<context>' )`, and every text of PHP through `ezpI18n::tr()`.

    - The extension ships `translations/<locale>/translation.ts` and names itself in `site.ini.append.php` (`[RegionalSettings] TranslationExtensions[]=<name>`).
    - The administration is translated into 156 languages. An extension of the product ships at least English and German.

  Reference: Translations and languages, Languages.

- **Help pages.** Keep `README.md` and `INSTALL.md` on top of `extension/<name>/` and every other page as Markdown (`.md`) in `extension/<name>/doc/`, in English. These pages, and the extension's `CHANGELOG*`, `LICENSE*`, `USAGE*` and `COPYING*`, appear in this help center as soon as the extension is active, with the page id `ext/<name>/<path>`. Only files the extension's repository tracks, or lists in its `share/filelist.md5`, are shown. Older extensions' plain-text pages still show, as plain text. How to: Documenting an extension; reference: The help system.

10. The website toolbar and Exponential Layouts

- **The website toolbar (expwt).** An extension adds a panel, a server action, a highlighted region or a search provider in its own `settings/expwt.ini.append.php`, under `[RegistrySettings]` (`PanelList[]`, `ActionList[]`) and a block `[Panel_<id>]` or `[Action_<extension>.<name>]` for each:
    - a panel names its label, its script in the extension's `design/<design>/javascript/` and who sees it (an expwt function and kernel policies);
    - an action names its class, which implements `expwtActionHandler` and checks the kernel policy of what it changes;
    - the toolbar never grants anything by itself.

  The full contract is in the header of `extension/expwt/classes/expwtregistry.php`.
- **Exponential Layouts.** Layout types, zones and block definitions are settings of `explayouts.ini.append.php` (`[LayoutType_*]`, `[BlockDefinition_*]`), and a block's look is a template of the extension's design. Reference: Layouts.

11. Tests and CI

- **Write tests that need no database.** The kernel's suite runs most tests without a database: plain arrays in, plain arrays out, with a fake for the database class where needed. CI runs them on every push, on PHP 8.1 to 8.5, and checks the syntax and boot on PHP 7.4 and 8.0.
- **Check before you push:** `php -l` for every changed PHP file, `xmllint --noout extension.xml`, then the tests of your extension.
- **The kernel's own CI**, how to reproduce a failure on your machine and the commit message policy: CI. PHPUnit versions and attributes: PHPUnit 13.

12. Release it

A release is a number written into the extension's files, a commit and a tag. **A published tag is permanent.** Packagist reads it once and caches it, so a tag is never moved, deleted or re-cut. A mistake is fixed by the next version.

1. **Choose the number from what is already published:** `git fetch --tags`, then `git tag -l 'v*' --sort=version:refname | tail -5`, and check the GitHub releases. Sort by version, not by name: after `0.1.9` comes `0.1.10`.
2. **Write it into both files** (`'Version'` in `ezinfo.php`, `<version>` in `extension.xml`, without the `v`) and into `composer.json` if it has a version. If the extension ships `share/filelist.md5`, refresh it in the same commit.
3. **Commit, then tag that commit**, push the branch and the tag, and create the release.
4. **Check Packagist.** If the new version does not appear, the repository's webhook to Packagist is missing.

Site packages only switch extensions on; they no longer carry copies of extensions (Site packages without extension packages). The whole release walk-through: Extensions, section 7, and Packages.

13. Troubleshooting

Symptom	Cause and fix
`Class '...' not found`	The autoloads are stale: `php bin/php/ezpgenerateautoloads.php -e`, then `./console exp:velocity deploy`. `php bin/php/checkclasses.php --quiet-ok` names any class that cannot load.
The module's address answers "module not found"	The extension is not active, or `settings/module.ini.append.php` lacks `ExtensionRepositories[]=<name>`.
The page is empty, or shows the layout without content	The template is not found: check the path below `design/<design>/templates/`, `DesignExtensions[]=<name>`, and clear the template-override cache after adding a new `.tpl`.
It works under Apache but not on Velocity's port	Velocity's workers keep the classes they loaded at start: deploy with `./console exp:velocity deploy` (it restarts Velocity).
A setting of the extension has no effect	Another extension earlier in the list, or `settings/override/`, sets it too: `./console exp:ini where <file>/<block>/<setting>` names the winner.
The extension's tables are missing	Run `./console exp:upgrade`; the step `6.0.15-active-extension-tables` creates them from `share/db_schema.dba`.
Its link is missing from the menu	The user lacks a policy of `PolicyList_<key>[]`, or the menu ends in `CollapseAfter` and the link is under **More**.
A new stylesheet or script has no effect	Deploy with `--packer`, then reload the page without the browser cache.
The about page shows no version	`ezinfo.php` and `extension.xml` disagree, or `extension.xml` lacks `<metadata><version>`.
The health check on `/setup/rad` lists a problem	It names the setting and the class that cannot work; fix the class name or regenerate the autoloads.

More:

- Extensions, the step-by-step guide to finding, building and releasing extensions;
- the book Extension development;
- Repairing an installation;

- Integrating With Exponential 6.0, for connecting other systems through the APIs instead of extending the code.

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.