Validation: input, datatypes and escaping

Where validation happens in Exponential when there is no validator component - content is validated by its datatypes when editors save, your own forms and parameters are validated in your view or service, and output is...

Source file: `doc/eng-us/help/web-framework-developers/validation.md` Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/web-framework-developers/validation.md

This page is for developers who declare constraints (Symfony Validator attributes, Laravel validation rules, Django form fields, Rails model validations). Exponential has no general validator component with declarative rules. Validation happens in three places, and knowing which one applies saves a lot of duplicate code.

Contents

1. Three places
2. Content: the datatypes validate
3. Your input: validate in the view or service
4. Showing errors
5. Output: escape where you print
6. Compared with validation libraries

Three places

Data	Validated by	You write
Content edited in the administration or a front-end edit form	the datatype of each attribute, with the class attribute's settings (required, length, ranges)	nothing, or a custom datatype
Parameters and forms of your own views	your view or the service it calls	casts, allow lists, checks, error messages
API input	the REST layer for content; your controller or server function for the rest	the same checks as for forms

Content: the datatypes validate

When content is saved through the edit views, each attribute's datatype checks the input in `validateObjectAttributeHTTPInput()` (`kernel/classes/ezdatatype.php`) and answers with an `eZInputValidator` state: `STATE_ACCEPTED`, `STATE_INTERMEDIATE` or `STATE_INVALID`, with a message the edit form shows next to the field. Whether a field is required, its length limits and ranges are settings of the class attribute, set in the class editor. A new kind of rule is a new datatype (Custom datatypes, The datatype life cycle).

When code creates content with `eZContentFunctions::createAndPublishObject()`, values are given in the datatype's `fromString()` form; validate them in your import code before you call it.

Your input: validate in the view or service

From Input validation and output escaping:

Input	Validate
IDs and numbers	`(int)`, then a range check (`max( 1, min( 100, (int)$limit ) )`)
Choices	an allow list: `in_array( $sort, array( 'name', 'published' ), true )`
Text	length limits, trim, the characters allowed for the field
E-mail, URL	`filter_var()`; for URLs also the scheme (`https` only)
Files	type by content and size, stored through the kernel
Redirect targets	only addresses of the site (Safe redirects)

Address parameters (`$Params`) are strings from the URL, as untrusted as POST data. Put the rules in a small validator class your view, command and API share, and test it with PHPUnit without a database (Testing).

PHP

```php
$errors = array();
$title = trim( (string)$Module->actionParameter( 'Title' ) );
if ( $title === '' )
    $errors['title'] = ezpI18n::tr( 'extension/expdemo', 'Enter a title.' );
elseif ( mb_strlen( $title ) > 200 )
    $errors['title'] = ezpI18n::tr( 'extension/expdemo', 'The title is too long.' );
```

Showing errors

Re-render the form with the values the user typed, show each error next to its field and a summary at the top, and link the summary to the fields so keyboard and screen reader users find them (Forms, CSRF protection and HTTP input):

PHP

```php
$tpl->setVariable( 'errors', $errors );
$tpl->setVariable( 'values', array( 'title' => $title ) );
```

Output: escape where you print

Templates do not escape automatically: every value from users or the database goes through `|wash`, except values that are already safe HTML by design (`$module_result.content`, `attribute_view_gui` output, rendered rich text) (The view layer). In PHP, use `htmlspecialchars( $v, ENT_QUOTES, 'UTF-8' )`, `json_encode()`, `$db->escapeString()` for their contexts.

Compared with validation libraries

Library feature	Exponential
Declarative constraints on a model	class attribute settings for content; nothing for your own tables
Validation groups, cascading	no

Library feature	Exponential
Translated messages	`ezpI18n::tr()` in your code; datatypes ship translated messages
Form error rendering	your template

A validation library from Composer can be used inside your extension if you prefer declarative rules; the kernel does not depend on one.

Related pages

- Forms and CSRF protection
- Datatype input hardening
- Security for applications
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

> **Become an Exponential Partner**
>
> Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.
>
> Partners: start here | How to apply

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