Errors and debugging

The messages the template engine logs - unknown variable, unknown attribute, unknown operator or function, parser errors with line and column, a missing template, def/set/undef misuse - each produced on a development...

Source file: `doc/eng-us/help/template-language/errors-and-debugging.md` Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/template-language/errors-and-debugging.md

The template engine never stops a page for a template mistake. It logs an error or a warning, outputs nothing (or the rest of the text) for the faulty part and goes on. This page shows the messages it produces, each one triggered on a development installation by a small template, so you can recognize them in the debug output, and the settings that help find the template behind a piece of HTML.

Contents

1. Where the messages go
2. Variables and attributes
3. Operators and functions
4. Parser errors
5. Missing templates
6. def, set and undef
7. The whitespace rule
8. Debug settings
9. Related pages

Where the messages go

`eZTemplate::error()` and `eZTemplate::warning()` (`lib/eztemplate/classes/eztemplate.php`) keep each message in the engine's error or warning log and pass it to `eZDebug` with the template file, line and column. You see them

- in the debug output of the page, when `[DebugSettings] DebugOutput=enabled` and your IP or user is allowed (`DebugByIP`, `DebugByUser`);
- in `var/log/error.log` and `var/log/warning.log` when the debug log for that level is enabled;
- on the page itself only with `template.ini [ControlSettings] DisplayWarnings=enabled` (disabled by default and on a development installation), which adds a warning box for template errors.

The examples below ran through the engine on a development installation; each shows the output and then what the engine logged. The file paths in the parser messages are the scratch paths the examples were kept in.

Variables and attributes

A variable that does not exist outputs nothing and logs an error with the namespace it was looked for in:

Template

```
before [{$no_such_variable}] after
```

Output:

```
before [] after
```

Logged by the template engine:

```
ERROR : Unknown template variable 'no_such_variable' in namespace ''
```

An attribute that an object or array does not have: the message names the class (or the array size) and lists what exists:

Template

```
{def $list = array( 'a', 'b' )}
[{$list.5}] [{$list.first}]
```

Output:

```
[] []
```

Logged by the template engine:

```
ERROR : No such attribute for array(2): 5
Choose one of following: 0, 1
ERROR : No such attribute for array(2): first
Choose one of following: 0, 1
```

For an object the message is `No such attribute for object(<class>): <name>`; see The object model in templates. Use `is_set()` to test without a message.

Operators and functions

An operator that is not registered on this siteaccess logs a warning and **passes its input through unchanged**:

Template

```
before [{'text'|no_such_operator}] after
```

Output:

```
before [text] after
```

Logged by the template engine:

```
WARNING : Operator 'no_such_operator' is not registered
```

An unknown function tag logs a warning, and nothing of the tag is output. Note that the text after the tag on the same line was not output either:

Template

```
before {no_such_function param=1} after
```

Output:

```
before
```

Logged by the template engine:

```
WARNING : Function "no_such_function" is not registered
```

Both happen when an extension that registers the operator or function is not active on the siteaccess, or when its autoload registration is missing (see Writing template operators).

Parser errors

A tag without its closing brace: the parser logs where the tag starts (`file:line[column]`) and outputs the rest of the template as text:

Template

```
line one
{def $a = 1
line three
```

Output:

```
line one
{def $a = 1
line three
```

Logged by the template engine:

```
ERROR : parser error @ tm-errors/04-unterminated-tag.tpl:1[0]
Unterminated tag, needs a } to end the tag.
{def $a = 1
line three
```

Extra characters in a function's parameters: the parser logs what it expected; the part it understood still runs (`$a` was set to 1):

Template

```
{def $a = 1 2}
[{$a}]
```

Output:

```
[1]
```

Logged by the template engine:

```
ERROR def: parser error @ tm-errors/05-extra-characters.tpl:1
($) expected at [2]
```

A block that is never closed is not reported at all: the parser treats the rest of the file as the block's body.

Template

```
{if true()}
open if, never closed
```

Output:

```
open if, never closed
```

Missing templates

An `include` (or a view function) whose template is not found in any design base logs a warning and outputs nothing:

Template

```
before [{include uri='design:no/such/template.tpl'}] after
```

Output:

```
before [] after
```

Logged by the template engine:

```
WARNING : No template could be loaded for "no/such/template.tpl" using resource "design"
```

For the search order behind "could not be loaded", see Template resources and overrides.

def, set and undef

`def` of an existing variable warns and assigns anyway; `set` of a variable that does not exist warns and does nothing; `undef` of a variable not made with `def` warns:

Template

```
{def $a = 1}
{def $a = 2}
{set $b = 3}
{undef $c}
[{$a}]
```

Output:

```
[2]
```

Logged by the template engine:

```
WARNING def: Variable 'a' is already defined.
WARNING set: Variable 'b' doesn't exist, cannot set
WARNING undef: Variable 'c' is not defined with {def}.
```

The rules are in `lib/eztemplate/classes/eztemplatedeffunction.php` and `eztemplatesetfunction.php`; the designer's page is Variables and def.

The whitespace rule

Many "missing line break" questions come from one rule of the parser (`eZTemplateMultiPassParser::parseWhitespaceRemoval()`, `lib/eztemplate/classes/eztemplatemultipassparser.php`): **one line break directly before and one directly after every function tag** (`{if}`, `{/if}`, `{foreach}`, `{def ...}`, `{include ...}` and the rest) is removed. Output tags such as `{$x}` or `{$x|wash}` keep the line breaks around them. Consequences:

- A function tag alone on its line leaves no empty line behind, which is what makes templates readable.
- Text between two function tags on separate lines loses both line breaks. To keep one line break between `{/if}` and the next `{if}`, leave an empty line between them; two empty lines keep two.
- A line that ends with a function tag and the line after it are joined.

The Data map and content examples use empty lines for exactly this reason.

Debug settings

All in `site.ini [TemplateSettings]` unless noted; values on the `site` siteaccess:

Template

```
Debug:                    {ezini( 'TemplateSettings', 'Debug' )}
ShowXHTMLCode:            {ezini( 'TemplateSettings', 'ShowXHTMLCode' )}
ShowUsedTemplates:        {ezini( 'TemplateSettings', 'ShowUsedTemplates' )}
ShowTemplatePathComments: {ezini( 'TemplateSettings', 'ShowTemplatePathComments' )}
DebugOutput:              {ezini( 'DebugSettings', 'DebugOutput' )}
DisplayWarnings:          {ezini( 'ControlSettings', 'DisplayWarnings', 'template.ini' )}
```

Output:

```
Debug:                    disabled
ShowXHTMLCode:            enabled
ShowUsedTemplates:        disabled
ShowTemplatePathComments: disabled
DebugOutput:              enabled
DisplayWarnings:          disabled
```

Setting	Effect
`Debug`	adds `<!-- START: including template: <file> (<uri>) -->` and STOP comments around every template that is parsed (not around a pagelayout that starts with `<!DOCTYPE`), and logs which files are loaded
`ShowXHTMLCode`	with Debug enabled, also prints the template path as visible text in the page

Setting	Effect
`ShowUsedTemplates`	records every template used; the debug output (and the debug bar's template list) shows them. Needs `DebugOutput`
`ShowTemplatePathComments`	wraps every rendered template in `<!-- START <path> -->` / `<!-- STOP <path> -->` comments, with the compiler on or off; development only, never on siteaccesses that serve JSON, XML or RSS
`ShowMethodDebug`	logs the start and end of every function and operator call (interpreted templates only)
`template.ini [ControlSettings] DisplayWarnings`	shows template errors as a warning box on the page

`Debug` comments are added when a template is parsed, so compiled templates that already exist do not get them until the template cache is cleared. `ShowTemplatePathComments` works with compiled templates as they are.

To find the file behind a piece of HTML on a running site, `ShowTemplatePathComments` is usually the quickest: enable it for one siteaccess in a development installation, view the page source, and search for the HTML. The Template debugging page walks through that.

Related pages

- Template debugging
- Template path comments
- Common template mistakes
- The template compiler
- Debugging and logs
- 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.