Namespaces

How the template engine keeps variables apart - the root and current namespace, the four ways to name a variable ($var, $ns:var, $:var, $#var), what include does with and without name=, the namespaces section and let...

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

Every template variable lives in a namespace, a name such as `''` (the top of the page), `box` or `Outer:Inner`. The engine stores variables as `$this->Variables[<namespace>][<name>]` (`lib/eztemplate/classes/eztemplate.php`), and every template is processed with two namespaces at hand: the **root namespace** of the file and the **current namespace**, which a `section` or `let` with a name can change inside the file. Namespaces are what keeps the variables of included templates and nested loops from overwriting each other.

Contents

1. Four ways to name a variable
2. include without and with a name
3. Namespaces made by section and let
4. A variable in the wrong namespace
5. Practical rules
6. Related pages

Four ways to name a variable

The variable parser (`variableEndPos()` in `lib/eztemplate/classes/eztemplateelementparser.php`) reads an optional scope sign, any number of `name:` prefixes and the variable name. `elementValue()` in `eztemplate.php` then finds the namespace:

Written	Scope	Namespace looked up
`$title`	local	the root namespace of the file
`$box:title`	local	root namespace + `:box` (or box at the top)
`$:title`	relative	the current namespace
`$#title`	global	exactly the namespace written, here `''`, the top of the page
`$#box:title`	global	exactly box

At the top of a page all of these are the same namespace `''`. They differ inside included templates, sections and `let` blocks.

include without and with a name

`eZTemplateIncludeFunction::process()` decides the namespace of the included file:

- **Without** `name`, the file runs in the caller's current namespace. It sees every variable of the caller, and the parameters it is given are set in that namespace for the time of the include.
- **With** `name='box'`, the file runs in the namespace `box` (or `<current>:box`). Its `$title` means `box:title`, so it sees only its parameters; the caller's variables are still reachable with `$#name`.

This file is included twice below:

Template

```
title=[{$title}] local=[{if is_set( $node )}{$node.name}{else}no $node{/if}] global=[{$#node.name
}] relative=[{$:title}]
```

Without a name it sees `$node` of the caller:

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 89 ) )}
{include uri='file:var/tmp/examples/tl-ns/parts/show.tpl' title='A'}
```

Output:

```
title=[A] local=[Fit & Healthy] global=[Fit & Healthy] relative=[A]
```

With `name='box'` the local `$node` is missing, the global `$#node.name` still works, and the parameter `$title` is removed again after the include:

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 89 ) )}
{include uri='file:var/tmp/examples/tl-ns/parts/show.tpl' title='B' name='box'}
after: [{if is_set( $title )}title leaked{else}no $title here{/if}]
```

Output:

```
title=[B] local=[no $node] global=[Fit & Healthy] relative=[B]
after: [no $title here]
```

Both forms appear in shipped designs. Without a name is convenient but couples the part to its callers; a name makes the part self-contained.

Namespaces made by section and let

`section` with `name=` processes its body in the namespace `<current>:<name>` and sets `item`, `key`, `index`, `number` (and `sequence`) there (`eZTemplateSectionFunction::process()`). Nested sections nest the names, and `$:item` always means the innermost:

Template

```
{section name=Outer loop=array( 'x', 'y' )}{section name=Inner loop=array( 1, 2 )}[{$Outer:item}{
$Outer:Inner:item}|{$:item}]{/section}{/section}
```

Output:

```
[x1|1][x2|2][y1|1][y2|2]
```

`let` and `default` with `name=` do the same for their variables:

Template

```
{let name=Box width=3}[{$Box:width}] [{$:width}]{/let}
```

Output:

```
[3] [3]
```

`foreach`, `for`, `while` and `def` do not create namespaces; their variables are ordinary variables of the file.

A variable in the wrong namespace

A lookup that misses names the namespace it searched, which tells you whether the variable is missing or only out of reach:

Template

```
{def $title = 'root'}
[{$Box:title}]
```

Output:

```
[]
```

Logged by the template engine:

```
ERROR : Unknown template variable 'title' in namespace 'Box'
```

Practical rules

- In new templates use `def` and `foreach`; they need no namespaces at all.
- Give a reusable part a `name` when you include it, and pass everything it needs as parameters.
- Reach for `$#var` only to read page-wide values from inside a named part; to pass values up, see set with a scope.
- When reading old templates, `$Name:item` is a `section` or `let` namespace: see Sections versus foreach.

Related pages

- Previous: Variables in depth
- Next: Attribute access
- Include
- Section (legacy)

- 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.