The persistent variable, inside

How $persistent_variable travels from a node view template to the pagelayout - the kernel sets it to false, reads it back from the root namespace after rendering, stores it in content_info and in the content view cache...

Source file: doc/eng-us/help/template-language/persistent-variable-internals.md Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/template-language/persistent-variable-internals.md

A node view template runs before the pagelayout and, when the page comes from the view cache, does not run at all. The persistent variable is the one channel from the view template to the pagelayout that works in both cases. The designer's how-to is The persistent variable; this page shows the kernel code behind it.

Contents

1. The path through the kernel
2. Why scope='global'
3. What survives the view cache
4. Helpers in extensions
5. Related pages

The path through the kernel

All in `eZNodeviewfunctions::generateNodeViewData()` (`kernel/classes/eznodeviewfunctions.php`):

1. Before the view template runs: `$tpl->setVariable( 'persistent_variable', false );`
2. The template is fetched: `$Result['content'] = $tpl->fetch( 'design:node/view/' . $viewMode . '.tpl' );`
3. Afterwards the kernel reads the variable back from the **root namespace** and stores it:

PHP

```php
$contentInfoArray['persistent_variable'] = false;
if ( $tpl->variable( 'persistent_variable' ) !== false )
{
    $contentInfoArray['persistent_variable'] = $tpl->variable( 'persistent_variable' );
    $keyArray[] = array( 'persistent_variable', $contentInfoArray['persistent_variable'] );
    $res->setKeys( $keyArray );
}
```

4. `content_info` goes into `$Result`, so the pagelayout reads `$module_result.content_info.persistent_variable`.
5. The value also becomes the override key `persistent_variable`, which `override.ini` can match on.

The same kernel step can be replayed in the example renderer: define the variable as false in the root namespace (as step 1 does), include a template that sets it with `scope='global'`, then

read it back:

Template

```
{* the kernel does this before it fetches node/view/<viewmode>.tpl *}
{def $persistent_variable = false()}
{include uri='file:var/tmp/examples/tm-persistent/view-part.inc'}

after the view template: body_class={$persistent_variable.body_class}, hide_sidebar={$persistent_
variable.hide_sidebar|choose( 'false', 'true' )}
```

Output:

```
<article>...</article>

after the view template: body_class=article, hide_sidebar=true
```

The included part (`view-part.inc` beside the example) is:

Template

```
{* stands in for node/view/full.tpl: sets the variable in the global namespace *}
{set scope='global' persistent_variable=hash( 'body_class', 'article', 'hide_sidebar', true() )}
<article>...</article>
```

Why scope='global'

`set` without a scope looks for the variable in the template's **current namespace** (scope `relative`, `lib/eztemplate/classes/eztemplatesetfunction.php`). The node view template itself runs in the root namespace, so a plain `{set persistent_variable=...}` there would work. But a template that is included with a `name=`, or that runs through `node_view_gui` (namespace `NodeView`), `content_view_gui` (`ContentView`) or `attribute_view_gui` (`ContentAttribute`), has a namespace of its own (`kernel/common/eztemplateautoload.php` declares these), and there the variable does not exist:

Template

```
{def $persistent_variable = false()}
{include uri='file:var/tmp/examples/tm-persistent/view-part-local.inc' name=part}

included with name=part, without scope: {if $persistent_variable}set{else}still false{/if}


{include uri='file:var/tmp/examples/tm-persistent/view-part.inc' name=part}

included with name=part, with scope='global': {$persistent_variable.body_class}
```

Output:

```


included with name=part, without scope: still false
<article>...</article>

included with name=part, with scope='global': article
```

Logged by the template engine:

```
WARNING set: Variable 'part:persistent_variable' doesn't exist, cannot set
```

The included part used here sets the variable without a scope:

Template

```
{* the same assignment, without scope='global' *}
{set persistent_variable=hash( 'body_class', 'article' )}
```

`scope='global'` writes to the global namespace (`''`) from anywhere, which is where step 3 reads (`scope='root'` means the root namespace of the current template, which inside an include is not always the same). Use it always, so the assignment keeps working when the template is moved into an include.

Note what `set` does not do: it never creates a variable. Only `def` does, and a `def` of a name that already exists logs "already defined" and assigns anyway (`lib/eztemplate/classes/eztemplatedeffunction.php`).

What survives the view cache

`content_info`, with the persistent variable in it, is part of the `$Result` that the content view cache stores. On a cache hit `eZNodeviewfunctions::contentViewRetrieve()` returns the stored `$Result` and sets the override keys again from `content_info`, `persistent_variable` included. The view template does not run, but the pagelayout gets the value it set when the cache was made.

This is also the limit: the value is whatever the template set when the cache file was written. Anything that varies per visitor or per request must not go into it; it would be served to every later visitor until the cache expires.

Helpers in extensions

- The ezwebin extension's `ezpagedata_set( key, value )` and `ezpagedata_append( key, value )` (`extension/ezwebin/autoloads/ezpagedata.php`) call `ezjscPackerTemplateFunctions::setPersistentArray()` of ezjscore, which writes into the same variable; `ezpagedata()` in the pagelayout merges `content_info.persistent_variable` into its page data.
- The admin designs read keys such as `left_menu` and `extra_menu` from it (for example `design/admin/templates/pagelayout.tpl`).
- `design/standard/templates/page_head.tpl` writes one `<meta name>` tag per entry of `$site.meta`, and takes the content from the persistent variable when it has a key of the same name (a view template can set its own `description`, for example).

Related pages

- The persistent variable
- Module view templates
- Module result reference
- Variables and def
- Template caches
- 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.