The fetch and fetch_alias operators

fetch and fetch_alias from eZTemplateExecuteOperator - how a template reaches a module's function_definition.php through eZFunctionHandler and eZModuleFunctionInfo, parameters, results and failures, what the compiler...

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

Templates read data with `fetch`, which is an operator, not a function tag: it returns a value that you assign with `def` or pipe into other operators. `fetch` and `fetch_alias` are registered by `eZTemplateExecuteOperator` (`lib/eztemplate/classes/eztemplateexecuteoperator.php`) through `lib/eztemplate/classes/eztemplateautoload.php`. The fetch functions themselves (content `node`, `list`, `tree`, user `current_user` and so on) are documented one by one in the fetch function reference.

Contents

1. fetch
2. fetch_alias
3. How a fetch is resolved
4. When a fetch fails
5. Compiled fetches
6. Related pages

fetch

fetch

Calls one function of a module and returns its result.

Parameter	Type	Required	Default
`module_name`	string, such as `content`, `user`, `class`, `section`	yes	`false`
`function_name`	string, a key of the module's `$FunctionList`	yes	`false`
`function_parameters`	hash of the function's parameters	no	`array()`

Input: none. Returns whatever the function puts in the `result` key of its answer: an object, a list, a number, or `null` when the call fails. Generated reference: fetch.

The Recipes category on a development installation (node 144) and its children:

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 144 ) )
     $count = fetch( 'content', 'list_count', hash( 'parent_node_id', 144 ) )
     $first = fetch( 'content', 'list', hash( 'parent_node_id', 144,
                                              'limit', 3,
                                              'sort_by', array( 'name', true() ) ) )}
{$node.name|wash} (node {$node.node_id}, class {$node.class_identifier}) has {$count} children; t
he first three by name:

{foreach $first as $child}
- {$child.name|wash}

{/foreach}
```

Output:

```
Recipes (node 144, class ng_category) has 18 children; the first three by name:
- Almond French Toast
- Avocado Kale Pesto Grilled Cheese
- Gluten-Free Shrimp Pad Thai with Tofu
```

Other modules work the same way. The CLI run that rendered this page is not logged in, so `current_user` is the anonymous user:

Template

```
{def $user = fetch( 'user', 'current_user' )
     $classes = fetch( 'class', 'list', hash( 'class_filter', array( 'folder', 'ng_article' ) ) )
}
Current user in this CLI run: {$user.login|wash} (logged in: {$user.is_logged_in|choose( 'no', 'y
es' )})
Classes: {foreach $classes as $c}{$c.identifier} {/foreach}
```

Output:

```
Current user in this CLI run: anonymous (logged in: no)
Classes: folder ng_article
```

The parameter hash may be a variable built earlier:

Template

```
{def $params = hash( 'parent_node_id', 144, 'limit', 2, 'sort_by', array( 'name', false() ) )}
{foreach fetch( 'content', 'list', $params ) as $k}{$k.name|wash}; {/foreach}
```

Output:

```
Zucchini Zoodles with Ground Turkey Tomato Sauce; Vibrant Guacamole;
```

fetch_alias

fetch_alias

Calls a fetch by a short name defined in `fetchalias.ini`. The alias names the module, the function, which template parameters map to which function parameters (`Parameter[]`) and fixed values (`Constant[]`); a parameter given in the template wins over a constant of the same name, and parameters the alias does not map are passed on as they are (

`eZFunctionHandler::executeAlias()`).

Parameter	Type	Required	Default
`function_name`	string, a section of `fetchalias.ini`	yes	`false`
`function_parameters`	hash	no	`array()`

Input: none. Returns the result of the fetch behind the alias. Generated reference: fetch_alias. The kernel defines eight aliases (`object`, `news_list`, `by_identifier`, `comments`, `children`, `children_count`, `subtree`, `reviews`); every alias, its mapping and how to add your own are on Fetch aliases.

Template

```
{def $kids = fetch_alias( 'children', hash( 'parent_node_id', 144, 'limit', 2, 'sort_by', array(
'name', true() ) ) )
     $n = fetch_alias( 'children_count', hash( 'parent_node_id', 144 ) )}
children_count: {$n}
children: {foreach $kids as $k}{$k.name|wash}; {/foreach}
```

Output:

```
children_count: 18
children: Almond French Toast; Avocado Kale Pesto Grilled Cheese;
```

How a fetch is resolved

1. `eZTemplateExecuteOperator::modify()` passes the module name, function name and parameters to `eZFunctionHandler::execute()` (`lib/ezutils/classes/ezfunctionhandler.php`).
2. `eZFunctionHandler::moduleFunctionInfo()` creates one `eZModuleFunctionInfo` (`lib/ezutils/classes/ezmodulefunctioninfo.php`) per module and request, and `loadDefinition()` looks for `<module path>/<module>/function_definition.php` in every path of `eZModule::globalPathList()`. Those paths come from `eZModule::activeModuleRepositories()`: `[ModuleSettings] ModuleRepositories` of `module.ini` (`kernel` and `kernel/private/modules`), then the `modules/` folder of every active extension listed in `ExtensionRepositories`. The **first** file found is used, and it must define `$FunctionList`.
3. `execute()` looks up the function, checks that it has a `call_method` (class and method) and `parameters`, and calls the method of the function collection class with the parameters in the declared order. A missing optional parameter gets its declared default.
4. The method returns an array; its `result` key is what the template receives.

Writing such a function: Writing your own fetch function.

When a fetch fails

An unknown module, an unknown function and a missing required parameter all give `null` in the template. The reasons ("Cannot execute function ... no valid data", "No such function", "Missing parameter") go to the debug output through `eZDebug::writeError()`, not to the template's own error list, so a failing fetch is silent on the page:

Template

```
unknown module: [{fetch( 'no_such_module', 'list' )|is_null|choose( 'not null', 'null' )}]
unknown function: [{fetch( 'content', 'no_such_function' )|is_null|choose( 'not null', 'null' )}]
missing parameter: [{fetch( 'content', 'node' )|is_null|choose( 'not null', 'null' )}]
```

Output:

```
unknown module: [null]
unknown function: [null]
missing parameter: [null]
```

Test the result before using it (`{if $node}` or `is_null`), and look at the debug output while developing.

Compiled fetches

When the module and function names are constants, the compiler (`fetchTransform()`) resolves the function definition while compiling and writes code that calls the function collection's method directly, with the parameters in their declared order, instead of going through `eZFunctionHandler` at run time. The method call is guarded: if the class or method cannot be loaded by the running process, the result is `null` and an error is written to the debug output. A fetch with a variable module or function name stays a run-time call. All examples on this page gave the same output interpreted and compiled on a development installation.

Related pages

- Fetch functions: reading data in templates
- Fetch basics for designers and Fetch performance
- Fetch aliases
- The fetch function reference (generated)
- All template operators (generated)
- 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.