The object model in templates

How a template sees the PHP objects of the content model - node, content object, version, data map, content object attribute and its datatype content - and the attribute protocol (attributes, hasAttribute, attribute)…

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

Content templates do not work with database rows. They work with PHP objects of the kernel, and every dot in a variable path (`$node.object.data_map.title.content`) is a call into one of them. This page explains which objects a template meets, how they hang together, and the small protocol the template engine uses to read them. The tables of every attribute are on the reference pages linked at the end.

Contents

1. The chain from node to content
2. A walk down the chain
3. The attribute protocol
4. Arrays and objects use the same dot
5. Testing before reading
6. What a missing attribute does
7. Node or object?
8. Related pages

The chain from node to content

```
eZContentObjectTreeNode      a place in the content tree        $node
  .object  ──────────────►  eZContentObject                     the content item, all its version
s
       .current  ─────────►  eZContentObjectVersion             the published version
  .data_map  ─────────────►  array( identifier => eZContentObjectAttribute )
       .title  ───────────►  eZContentObjectAttribute           one field of the class
            .content  ────►  datatype content                   string, ezxmltext, eZImageAliasHa
ndler, array ...
```

- **Node** (`eZContentObjectTreeNode`, `kernel/classes/ezcontentobjecttreenode.php`): a location in the tree. One object can have several nodes (locations); each has its own parent, path, URL alias and children.
- **Content object** (`eZContentObject`, `kernel/classes/ezcontentobject.php`): the content itself, with its class, section, owner, states, languages and versions.
- **Version** (`eZContentObjectVersion`, `kernel/classes/ezcontentobjectversion.php`): one version of the object; `current_version` is the published one.
- **Content object attribute** (`eZContentObjectAttribute`, `kernel/classes/ezcontentobjectattribute.php`): one field of one version in one

language. `data_map` is the map from the class attribute identifier to these objects.

- **Content**: what the field's datatype makes of the stored data (`eZContentObjectAttribute::content()` asks the datatype's `objectAttributeContent()`). It is a plain string for `ezstring`, an `eZXMLText` object for `ezxmltext`, an `eZImageAliasHandler` for `ezimage`, an array for `ezobjectrelationlist`, and so on.

The node's `data_map` and the object's `data_map` normally hold the same attributes: `eZContentObjectTreeNode::dataMap()` (`kernel/classes/ezcontentobjecttreenode.php`) returns the data map of the node's object for the version stored on the node (`contentobject_version`), or the current user's latest draft while `eZContentObjectTreeNode::setUseCurrentUserDraft( true )` is switched on (off by default). `eZContentObject::fetchDataMap()` keeps each map in the global `$eZContentObjectDataMapCache`, so reading it twice in one request costs one query.

A walk down the chain

Node 96 on a development installation is an `ng_article` below "Fitness" in the Fit & Healthy site:

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 96 ) )}
node:      {$node|get_class} {$node.node_id}, parent {$node.parent_node_id}, depth {$node.depth},
 path {$node.path_string}
object:    {$node.object|get_class} {$node.object.id} "{$node.object.name|wash}", class {$node.ob
ject.class_identifier}
version:   {$node.object.current|get_class} {$node.object.current.version} of {$node.object.versi
ons|count}
data_map:  {$node.data_map|get_type}
attribute: {$node.data_map.title|get_class} {$node.data_map.title.id}, datatype {$node.data_map.t
itle.data_type_string}
content:   {$node.data_map.title.content|get_type} "{$node.data_map.title.content|wash}"
```

Output:

```
node:      ezcontentobjecttreenode 96, parent 92, depth 4, path /1/2/89/92/96/
object:    ezcontentobject 94 "Why Should You Choose Strength Training?", class ng_article
version:   ezcontentobjectversion 1 of 1
data_map:  array[18]
attribute: ezcontentobjectattribute 580, datatype ezstring
content:   string[40] "Why Should You Choose Strength Training?"
```

`get_class` returns the class name in lower case and `get_type` the type with its size (see the type operators). The article has one version, and its `title` field is an `ezstring` whose content is a plain string.

The attribute protocol

Every class the kernel hands to templates implements three methods, defined once in `eZPersistentObject` (`kernel/classes/ezpersistentobject.php`):

Method	What it does
`attributes()`	the names a template may read: the database `fields` of `definition()`, then its `function_attributes`, then its `functions`

Method	What it does
`hasAttribute( $name )`	true when the name is one of those
`attribute( $name )`	a function attribute first: calls the PHP method named in `function_attributes`; else the member variable named in `fields`; else a `functions` method; else logs "Attribute '...' does not exist" and returns null

So `$node.name` is not the database column: `eZContentObjectTreeNode::definition()` maps `name` to the function attribute `getName()`, and `$node.data_map` to `dataMap()`. When a name is both a field and a function attribute (`remote_id` on the node and on the object, `name` on the object), the function attribute wins, because `attribute()` checks `function_attributes` first.

The engine side is in `lib/eztemplate/classes/eztemplate.php`: for each dot it checks `is_object()` and then calls `hasAttribute()` and `attribute()`; for an array it checks `isset( $array[$key] )`. Any PHP class that implements `attributes()`, `hasAttribute()` and `attribute()` can be handed to a template and read with dots; that is how extensions expose their own objects. The compiled form of a template does the same through `compiledFetchAttribute()` (see The template compiler).

A function attribute runs PHP every time it is read. `$node.children` fetches all children, `$node.object.versions` loads all versions; read such attributes once into a variable with `def` instead of in a loop.

Arrays and objects use the same dot

A dot reads an array key or an object attribute alike. A number reads a list element, and square brackets take an expression, so the key can come from a variable:

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 96 ) )
     $path = $node.path
     $field = 'title'}
{$path|count} ancestors: {$path.0.name|wash}, then {$path[1].name|wash}, then {$path[2].name|wash
}
path_array: {$node.path_array|implode( '/' )}
dynamic key: {$node.data_map[$field].content|wash}
```

Output:

```
3 ancestors: Websites, then Fit &amp;amp; Healthy, then Fitness
path_array: 1/2/89/92/96
dynamic key: Why Should You Choose Strength Training?
```

`$node.path` holds the ancestors as nodes (from the first level below the root, without the node itself), and `path_array` the node IDs of the whole path. The `&` in "Fit & Healthy" is `wash` escaping the ampersand for HTML.

Testing before reading

`is_set()` asks the same question without an error, so a template that serves several classes can test for a field:

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 96 ) )}
{if is_set( $node.data_map.image )}has an image field{/if}


{if is_set( $node.data_map.price )}has a price field{else}no price field{/if}
```

Output:

```
has an image field
no price field
```

What a missing attribute does

Reading an attribute that does not exist outputs nothing and logs an error naming the class. The value is null, so the template goes on:

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 96 ) )}
[{$node.no_such_attribute}]
```

Output:

```
[]
```

Logged by the template engine:

```
ERROR : No such attribute for object(eZContentObjectTreeNode): no_such_attribute
Choose one of following:
```

The message comes from `eZTemplate::elementValue()`. When debug output is enabled for your IP, it appears in the debug panel of the page, with the template file, line and column. See Errors and debugging for the other messages.

Node or object?

Both lead to the same content; which one you start from decides what else you can reach:

Template

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 96 ) )
     $object = fetch( 'content', 'object', hash( 'object_id', $node.contentobject_id ) )}
by node:   {$node.name|wash} / main node {$node.object.main_node_id}
by object: {$object.name|wash} / main node {$object.main_node_id} / {$object.assigned_nodes|count
} location(s)
same data_map: {if eq( $node.data_map.title.id, $object.data_map.title.id )}yes{else}no{/if}
```

Output:

```
by node:   Why Should You Choose Strength Training? / main node 96
by object: Why Should You Choose Strength Training? / main node 96 / 1 location(s)
same data_map: yes
```

- Start from the **node** for anything about the place in the tree: parent, children, path, URL

alias, visibility, sort order, the `can_*` checks of that location.

- Start from the **object** for anything about the content item: class, section, owner, states, versions, languages, relations, all its locations (`assigned_nodes`).
- `content_view_gui` takes an object, `node_view_gui` a node (see Node view gui and Content view gui).

Related pages

- Node attributes reference
- Content object attributes reference
- Content attribute and version reference
- Data map and content
- Nodes, objects and attributes (the designer's introduction)
- Attributes and dot notation
- The content API (the same objects from PHP)
- 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.