Content from PHP

For CMS developers who read and write content in PHP scripts, importers and integrations with the real kernel API. It covers running a script with the kernel loaded, fetching nodes and lists, reading attributes...

Source file: doc/eng-us/help/cms-developers/content-from-php.md Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/cms-developers/content-from-php.md

This page is for developers who need content in PHP: an importer, a nightly sync with another system, a report, a custom module. The kernel has a supported way to do each of these, and using it is what keeps versions, workflows, search and caches right. Never write to the content tables directly.

Contents

1. Running a script with the kernel loaded
2. Reading a node and its object
3. Lists and subtrees
4. Reading attribute values
5. Creating content
6. Updating content
7. The user a script runs as
8. Moving, hiding and removing
9. Large runs

Running a script with the kernel loaded

The quickest way runs any PHP file with the bootstrap done, from the installation root:

Shell

```
php bin/php/ezexec.php extension/exp_mysite/scripts/report.php
```

For a script you keep, with options, help and progress output, write an `eZScript` script or a console command in your extension (eZScript and eZCLI, CLI scripts and console commands). The lesson A script with the kernel loaded walks through a first one.

Reading a node and its object

PHP

```php
<?php
$cli = eZCLI::instance();
$node = eZContentObjectTreeNode::fetch( 2 );
if ( !$node instanceof eZContentObjectTreeNode )
{
    $cli->error( 'Node 2 was not found' );
    return;
}
$object = $node->object();
$cli->output( $node->attribute( 'name' ) . ' (' . $node->attribute( 'class_identifier' ) . ')' );
$cli->output( 'Object ' . $object->attribute( 'id' ) . ', main node ' . $object->attribute( 'main
_node_id' ) );

$imported = eZContentObject::fetchByRemoteID( 'exp-import-event-42' );
```

`eZContentObjectTreeNode::fetch()` takes a node ID; `eZContentObjectTreeNode::fetchByRemoteID()`, `fetchByURLPath()` and `eZContentObject::fetch()` / `fetchByRemoteID()` find content by other keys (`kernel/classes/ezcontentobjecttreenode.php`, `kernel/classes/ezcontentobject.php`).

Lists and subtrees

`subTreeByNodeID()` is the PHP side of the `list` and `tree` fetches, with the parameters in CamelCase:

PHP

```php
<?php
$params = array( 'Depth' => 1, 'DepthOperator' => 'eq',
                 'ClassFilterType' => 'include', 'ClassFilterArray' => array( 'exp_event' ),
                 'SortBy' => array( 'published', false ),
                 'Offset' => 0, 'Limit' => 20, 'LoadDataMap' => true );
$events = eZContentObjectTreeNode::subTreeByNodeID( $params, $eventsFolderNodeID );
$total = eZContentObjectTreeNode::subTreeCountByNodeID( $params, $eventsFolderNodeID );
```

Other parameters: `AttributeFilter`, `ExtendedAttributeFilter`, `MainNodeOnly`, `IgnoreVisibility`, `Language`, `OnlyTranslated`, `Limitation`, `AsObject`. Access policies and visibility apply as for the current user unless you switch them off; `'Limitation' => array()` skips the access check and is only for code that checks access itself (eZContentObjectTreeNode).

Reading attribute values

PHP

```php
<?php
$dataMap = $node->dataMap();                // attribute identifier => eZContentObjectAttribute
if ( isset( $dataMap['title'] ) && $dataMap['title']->hasContent() )
{
    $title = $dataMap['title']->content();   // the datatype's value
    $asString = $dataMap['title']->toString(); // the same as a string, in the import format
}
```

`content()` returns what the datatype gives templates as `.content` (a string, an `eZDateTime`, an image handler, an array with `relation_list` ...). `toString()` returns the format `fromString()` reads, which is what you want for exports (The eZDataType API).

Creating content

`eZContentFunctions::createAndPublishObject()` (`kernel/classes/ezcontentfunctions.php`) creates the object, its first version and its node, and runs the publish operation:

PHP

```php
<?php
$object = eZContentFunctions::createAndPublishObject( array(
    'class_identifier' => 'exp_event',
    'parent_node_id'   => $eventsFolderNodeID,
    'remote_id'        => 'exp-import-event-42',
    'attributes'       => array(
        'title'      => 'Opening of the new hall',
        'intro'      => 'Doors open at ten.',
        'start_date' => (string)mktime( 10, 0, 0, 11, 2, 2026 ),
    ),
) );
if ( !$object instanceof eZContentObject )
{
    eZDebug::writeError( 'Could not create the event', __METHOD__ );
}
```

Values are strings in each datatype's `fromString()` format: text as it is, `1/0` for a checkbox, a timestamp for dates, a file path for images and files (with `storage_dir`), the stored XML for rich text. Always set `remote_id` on imported content, so the next run updates instead of duplicating (Creating, publishing and changing content from PHP).

Updating content

PHP

```php
<?php
$object = eZContentObject::fetchByRemoteID( 'exp-import-event-42' );
if ( $object instanceof eZContentObject )
{
    eZContentFunctions::updateAndPublishObject( $object, array(
        'attributes' => array( 'intro' => 'Doors open at eleven.' ),
    ) );
}
```

Only the attributes you pass change. `language` adds or updates a translation of the same object (Translations and languages). Each call creates and publishes a new version, with its workflows and notifications; pass `false` as the third argument to publish without notification events.

The user a script runs as

A command-line script runs as the anonymous user unless it sets one, and the anonymous user usually may not create content. Use a dedicated importer account with exactly the rights the import needs, not an administrator:

PHP

```php
<?php
$user = eZUser::fetchByName( 'exp_importer' );
if ( $user instanceof eZUser )
{
    eZUser::setCurrentlyLoggedInUser( $user, $user->attribute( 'contentobject_id' ) );
}
```

The created objects get this user as owner.

Moving, hiding and removing

Task	Call
Move a node	`eZContentObjectTreeNodeOperations::move( $nodeID, $newParentNodeID )`
Hide or show a subtree	`eZContentObjectTreeNode::hideSubTree( $node )`, `unhideSubTree( $node )`
Remove nodes (to the trash by default)	`eZContentObjectTreeNode::removeSubtrees( $nodeIDs, $moveToTrash = true, $infoOnly = false )`

`removeSubtrees()` with `$infoOnly = true` reports what would be removed: use it for a dry run.

Large runs

- Clear the in-memory object cache now and then (`eZContentObject::clearCache()`), or memory grows with every object.
- Offer a `--dry-run` that reports what would be created or changed.
- Run on a copy first, then on live in a quiet hour; large changes can also run as content jobs in the background (Content jobs).
- For the import tools that already exist, read Importing content before writing code.

Related pages

- Lesson 4.3: Reading and writing content from PHP
- eZContentObject
- Fetching content in templates
- 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.