Static state traps

An explanation of the bugs that come from state kept between requests or tests - function statics, register-once flags, statics assigned outside the class body, per-request caches keyed by time, settings and singletons...

Source file: `doc/eng-us/help/php-developers/static-state-traps.md` Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/php-developers/static-state-traps.md

This page is for PHP developers who keep anything in static properties, function statics or globals. The Exponential kernel uses static state a lot: singletons (`eZINI::instance()`, `eZDB::instance()`, `ezpEvent::getInstance()`) and in-memory caches. Under PHP-FPM that state disappears with the request. Three situations keep it alive: a Velocity worker that serves many requests, a long command line run, and a PHPUnit run with many tests in one process. Each of the traps below has hit the kernel itself.

Contents

1. Where state survives
2. Trap 1: function statics
3. Trap 2: register-once flags
4. Trap 3: statics assigned outside the class
5. Trap 4: caches keyed by too little
6. Trap 5: values captured at warm-up
7. Trap 6: caches that only grow
8. How to find them

Where state survives

Situation	Kept between	Reset
Velocity worker	requests of different users, siteaccesses, languages	class statics back to their post-warm-up values; shutdown functions and handlers per request (Velocity persistent workers for PHP developers)
a command that loops over many items	iterations	nothing, unless you clear
a cronjob run	the scripts of one part	nothing
PHPUnit	tests	nothing, unless the test resets

Trap 1: function statics

PHP

```php
// wrong: survives in a worker, and nothing resets function statics the way class statics are res
et
function acmeAssetPaths()
{
    static $paths = null;
    if ( $paths === null )
        $paths = acmeComputePathsForCurrentSiteaccess();
    return $paths;
}
```

The kernel had exactly this: function statics carried one siteaccess's asset paths into the next request. The fix was to move them into class statics, which the worker resets, and to key the cache by what it depends on (Trap 4).

Trap 2: register-once flags

PHP

```php
// wrong: the flag is set during the warm-up and stays true; later requests register nothing
if ( !self::$registered )
{
    self::$registered = true;
    register_shutdown_function( array( __CLASS__, 'flush' ) );
}

// right: once per request
if ( self::$registered !== eZExecution::requestKey() )
{
    self::$registered = eZExecution::requestKey();
    register_shutdown_function( array( __CLASS__, 'flush' ) );
}
```

The opposite failure exists too: a listener attached on every request without a guard ran twice. Make registrations idempotent, or put listeners into settings.

Trap 3: statics assigned outside the class

PHP

```php
// wrong
class acmeFooServices extends expServiceBase
{
    public static $services = array();
}
acmeFooServices::$services = array( 'bar' => array( ... ) );   // runs only when the file is firs
t included

// right
class acmeFooServices extends expServiceBase
{
    public static $services = array( 'bar' => array( ... ) );
}
```

A worker puts statics back to their **declared** value between requests, and `include_once` does not run the file-level assignment again. In the kernel's `expservices`, ten domains declared their service lists this way until 2026-10-12: from the second request on, every service of those classes answered "not declared" (404) on that worker (Backend services over ezjscore).

Trap 4: caches keyed by too little

A static cache must be keyed by everything the value depends on: the siteaccess, the language, the user or role, the request. Values cached under a key that ignores one of them leak between requests:

PHP

```php
$key = $GLOBALS['eZCurrentAccess']['name'] . '|' . $languageCode . '|' . $nodeID;
```

Per-request kernel caches use the request's key, which depends on the per-request `REQUEST_TIME`; that is why Velocity's engine release 0.0.4.42, which made `REQUEST_TIME` and `REQUEST_TIME_FLOAT` per request, was needed by the kernel (Velocity engine upgrade notes).

Trap 5: values captured at warm-up

Anything the warm-up computes is inherited by every worker: the warm-up once left the public site's language, its hidden-node setting and its template locations in every worker, until the kernel reset them. Do not compute request-dependent values in code that runs at class load or in the warm-up; compute them when a request needs them.

Trap 6: caches that only grow

In a long command, the kernel's in-memory caches (content objects, nodes, attributes) grow with every item:

PHP

```php
foreach ( $batches as $batch )
{
    acmeReportService::process( $batch );
    eZContentObject::clearCache();      // drop the in-memory object cache between batches
}
```

Your own static caches need a bound or a reset method that the loop calls.

How to find them

1. Under Velocity, request the same page twice in a row as two different users, in two siteaccesses, in two languages.
2. Compare with PHP's built-in server (`--engine=php`), which starts each request fresh: a difference is kept state.
3. In PHPUnit, run the suspect test file alone and together with others; a pair that fails only together shares state.
4. Search your code: `grep -rn "static \$" extension/acmereport/` and review each one with this page.

Related pages

- Velocity persistent workers for PHP developers
- Singleton pattern

- Registry pattern
- 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.