# The kernel service container

How PHP code asks the kernel for shared objects by name (expKernelServices::get()), how to define a service in kernelservices.ini with a class or factory, arguments, a scope and tags, what worker, request and prototype...

Source file: doc/eng-us/help/php-developers/the-kernel-service-container.md Last changed 2026-11-04
https://doc.exponential.earth/exphelp/viewer/help/php-developers/the-kernel-service-container.md

This page is for PHP developers who write kernel code or extensions and want to use, or offer, shared objects of the kernel. As of 6.0.15 the kernel has a service container: objects by name, defined in settings, made when first asked for, and kept in a scope. The contributor reference with every detail is the kernel service container.

## Contents

1. Asking for a service
2. Defining a service
3. Scopes
4. Request reset hooks
5. Compiling and checking
6. Testing

## Asking for a service

PHP

```php
$db = expKernelServices::get( 'db' );              // eZDB::instance()
$site = expKernelServices::get( 'ini.site' );      // eZINI::instance( 'site.ini' )
if ( expKernelServices::has( 'my.mailer' ) ) { /* ... */ }
foreach ( expKernelServices::tagged( 'kernel.core' ) as $id => $service ) { /* ... */ }
```

The existing calls such as `eZDB::instance()` and `eZINI::instance()` keep working and give the same objects; move to `expKernelServices::get()` when it is convenient. Every `ini.<name>` id is `eZINI::instance( '<name>.ini' )` and needs no definition.

## Defining a service

In `extension/<yours>/settings/kernelservices.ini.append.php` (or the kernel's `settings/kernelservices.ini`):

INI

```ini
[Service_my.mailer]
Class=myMailer
Arguments[]=@db
Arguments[]=%ini:mymailer.ini/MailSettings/Sender%
Arguments[]=plain text
Scope=request
Tags[]=my.senders
Description=The mailer of my extension.

[Tags]
TagList[]=my.senders
```

- `Class` is made with `new Class( ...Arguments );` with `Factory=myMailer::create` that static method is called instead.
- `@<id>` is another service, `%ini:<file>/<Section>/<Variable>%` a setting (null when it is not set), anything else is passed as written (`@@x` and `%%x` for a literal `@x` or `%x`).
- A later block of the same name wins, as everywhere in the settings; `Enabled=false` switches a service off.
- Nothing is found by reflection or scanning at runtime: the definition says everything.

## Scopes

| Scope | PHP-FPM, Apache | Velocity (persistent workers) | Use it for |
|---|---|---|---|
| `worker` | one per request | one per worker and siteaccess, kept for every request the worker serves | settings-like objects that hold **nothing of a request** |
| `request` (default) | one per request | one per request | anything that knows the visitor, the session, the URI or the database handle of the request |
| `prototype` | new each time | new each time | small value objects |

A worker-scoped service that keeps a request's data serves it to the next visitor on Velocity. `exp:services:check` refuses a worker-scoped service that takes a request-scoped one as an argument. When in doubt, use `request`; read static state traps and Velocity persistent workers for PHP developers.

For a truly dynamic object there is `expKernelServices::set( 'my.thing', $object )`: for the rest of this request only, never kept and never compiled.

## Request reset hooks

The web kernel starts every request with `expKernelServiceScope::resetRequest()`: the kernel's own resets (query cache, SQL profile, Content Security Policy nonce, audit, render statistics, request log and classifier), then yours:

PHP

```php
expKernelServiceScope::onRequestReset( 'my.extension', array( 'myCache', 'resetRequest' ) );
```

Registering the same name again replaces the hook, so registering on every request is fine; the hook is kept with the worker and runs at the start of each later request, on Velocity too.

## Compiling and checking

Shell

```
./console exp:services:compile      # also done by exp:velocity deploy and ezcache.php --clear-tag=ini
./console exp:services:check        # every siteaccess: definitions and compiled files, exit 1 on an error
./console exp:services:list --tag=my.senders -s site
./console exp:services:show my.mailer
```

The compiled container (`var/cache/kernelservices/`) also holds the recorded extension activation of each siteaccess, which the web kernel applies instead of looking up every extension directory: see the kernel service container for how it stays correct.

## Testing

DB-less tests need no compiled container: without one the container reads `kernelservices.ini` as it stands. Call `expKernelServices::reset()` in `tearDown()` when a test sets services or worker state. The kernel's own tests are in `tests/tests/kernel/classes/expKernelServicesTest.php` (PHPUnit without a database).

## Learn more

- The kernel service container (contributor reference)
- Reading settings with eZINI
- Write a console command
- How a request runs in PHP

> **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.