# Persistence cache

Persistence cache is a new feature introduced in 5.1. It does not exists in previous versions.



## Introduction

This page describes how Persistence cache works, and how to reuse the cache service it uses.

**Configuration**

For configuring the cache service, look at the Persistence cache configuration page.

## Persistent cache

### Layers

Persistence cache can best be described as a implementation of SPI\Persistence that wraps around the main implementation (currently: "Legacy Storage Engine").

As shown in the illustration this is done in the exact same way as the SignalSlot feature is a custom implementation of API\Repository wraps around the main Repository. In the case of Persistence Cache, instead of sending events on calls passed on to the wrapped implementation, most of the load calls are cached, and calls that perform changes purges the affected caches. This is done using a Cache service which is provided by StashBundle; this Service wraps around the Stash library to provide Symfony logging / debugging functionality, and allows configuration on cache handlers (Memcached, Apc, Filesystem, ..) to be configured using Symfony configuration. For how to reuse this Cache service in your own custom code, see below.

### Transparent cache

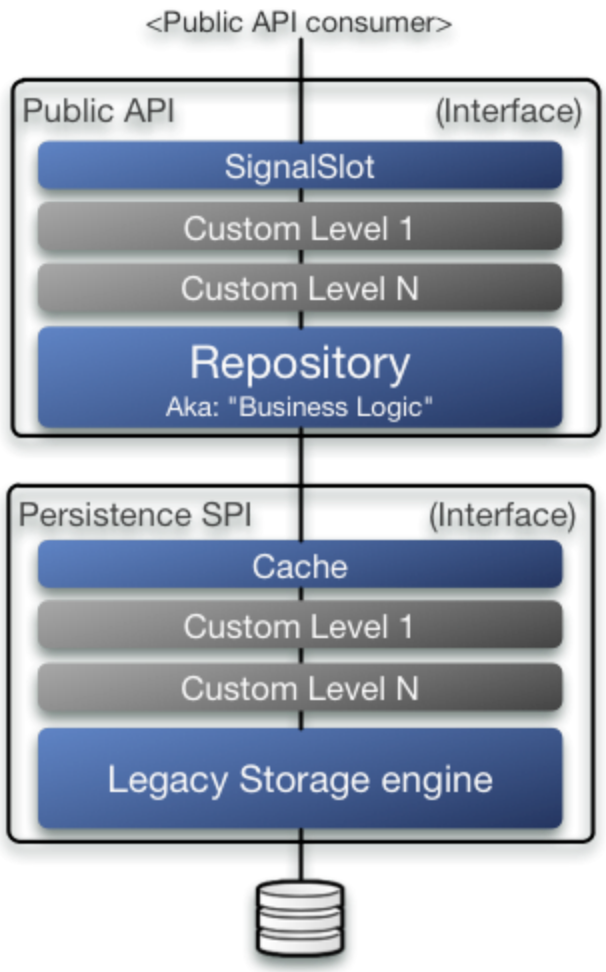
The persistence cache, just like the HTTP cache, tries to follow principles of "Transparent caching", this can shortly be described as a cache which is invisible to the end user and to the admin/editors of eZ Publish where content is always returned "fresh". In other words there should not be a need to manually clear the cache like was frequently the case with eZ Publish 4.x (aka "legacy"). This is possible thanks to an interface that follows CRUD (Create Read Update Delete) operations per domain, and number of other operations capable of affecting a certain domain is kept to a minimum.

#### Entity stored only once

To make the transparent caching principle as effective as possible, entities are as much as possible only stored once in cache by their primary id. Lookup by alternative identifiers (identifier, remoteId, ..) is only cached with the identifier as cache key and primary id as it's cache value, and com positions (list of objects) usually keep only the array of primary id's as their cache value.

This means a couple of things:

- Memory consumption is kept low
- Cache purging logic is kept simple ( Example: $sectionService->delete( 3 ) clears "section/3" cache entry )
- Lookup by identifier and list of objects needs several cache lookups to be able to assemble the result value




- Cache warmup usually takes several page loads to reach full as identifier is first cached, then the object

### What is cached?

Persistence cache aimed in its first iteration for caching all SPI\Persistence calls used in a normal page load, including everything needed for permission checking and url alias lookups.
However Search queries are currently not cached as it is more difficult to make sure they stay fresh unless all search cache is purged on every modification, or a complicated search cache walking purge system is implemented that is able to detect which search result to clear (this is what is planned for future versions, but it is planned to be done when there is a background process system in place).

Anyway, the following SPI calls are not currently cached:

- ObjectStateHandler
- TrashHandler
- UrlWildCardHandler
- SearchHandler::*
- transactions

For more details on which calls are cached or not, and where to contribute additional caches, check out the source.

### Legacy kernel cache purging

Currently with the Dual-kernel eZ Publish has in version 5.x, the "Transparent caching principle" referred to above has one major obstacle. eZ Publish 4.x ("legacy") kernel was not made for such a thing and has a lot of API's that can make changes to the data in the database and hence make the persistence cache "stale" (aka out of date).

A couple of things are in place to try to avoid this from happening / making it less of a problem:

- The Persistence cache has a expiry time configurable in ezpublish.yml
- LegacyBundle (the bundle that exposes legacy to Symfony and integrates Symfony with legacy) has a PersistenceCachePurger

#### PersistenceCachePurger

PersistenceCachePurger is setup by LegacyBundle to receive all relevant cache events triggered by legacy kernel, and clear relevant Persistence cache based on the incoming data.
This means a "Clear all cache" operation done in legacy will also clear all persistence cache, it also means relevant content cache is cleared on publishing, so all code using the api's covered by these events should in effect be cache safe in regards to persistence cache.

So, in case of stale persistence cache, a clear all cache in legacy admin interface is thus still possible, however a manual cache clear is also possible but how to do it depends on which cache handler is currently used.

## Reusing Cache service

Using the cache service allows you to use a interface  and not have to care about if the system has been configured to place the cache in Memcached, Apc or on File system. And as eZ Publish requries that instances uses a cluster aware cache, you can safly assume your cache is shared across all eZ Publish web servers.

> **Interface warning**
> Current implementation uses a caching library called Stash, via Stash-bundle. If this changes, then the Interface of the cache service will most likely change as well.

> **Cache key warning**
> When reusing the cache service within your own code, it is very important to not conflict with the cache keys used by eZ Publish, hence why example of usage starts with a unique "myApp" key for the namespace of your own cache, you must do the same!
>
> Also, this means that when eZ Publish clears all cache, it will also wipe out your custom cache as well!

**Multi site warning**
Be aware that in some edge cases as of eZ Publish 5.1, we recommend that the cache system is practically disabled using a *Blackhole* driver. This is the case in multi repository (one eZ Publish install, several databases) because the current configuration does not support setting cache key prefixes pr database (the current configuration is global to installation and not possible to set pr site / site group).

In this case the cache service will only keep your cache during the request, if the recommended InMemory setting is enabled.

# Get Cache service

## Via Dependency injection

In eZ Publish 5.x Symfony2 stack you can simply define that you require the "cache" service in your configuration like so:

**yml configuration**

```
myApp.myService:
        class: %myApp.myService.class%
        arguments:
            - @stash.default_cache
```

The "cache" service is an instance of the following class: Tedivm\StashBundle\Service\CacheService

## Via Symfony2 Container

Like any other service, it is possible to get the "cache" service via container as well like so:

**getting the cache service in php**

```
/** @var $cacheService \Tedivm\StashBundle\Service\CacheService */
$cacheService = $container->get( 'stash.default_cache' );
```

## In legacy via Symfony2 Container

When eZ Publish legacy runs via eZ Publish 5.x Symfony2 stack, you will be able to get the service container in the following way:

**Getting cache service in legacy**

```
// From a legacy module or any PHP code running in legacy context.
$container = ezpKernel::instance()->getServiceContainer();

/** @var $cacheService \Tedivm\StashBundle\Service\CacheService */
$cacheService = $container->get( 'stash.default_cache' );
```

# Using the cache service

Example usage of the cache service:

**Actuall example from cache use in ezpublish-kernel**

```
$cacheItem = $cacheService->getItem( 'myApp', 'object', $id );
    if ( $cacheItem->isMiss() )
    {
        $myObject = $container->get('my_app.backend_service')->loadObject( $id )
        $cacheItem->set( $myObject );
    }
    else
    {
        $myObject = $cacheItem->get();
    }
    return $myObject;
```

For more info on usage, take a look at Stash's documentation.