

Scope and Stores

Scope and Stores

Anonymous · 10/10/2026

Scope and Stores

Scope: Local Only

`LocalCacheBackend` and `LocalCacheFactory` cover exactly the backends that need no connection, no credentials, and no network round trip: they live in this process, in shared memory on this server, or on this filesystem.

```
enum LocalCacheBackend: string
{
    case None = 'none';           // Never caches anything (NullAdapter).
    case Memory = 'memory';       // Dies with the process (ArrayAdapter).
    case Apcu = 'apcu';           // Shared memory, survives requests (ApcuAdapter).
    case Filesystem = 'filesystem'; // Serialized files, supports objects
    (FilesystemAdapter).
    case PhpFiles = 'php_files';  // Plain PHP array files, OPcache-friendly
    (PhpFilesAdapter).
}
```

Redis, Memcached, Couchbase, PDO, and every other connection-based backend `symfony/cache` supports are deliberately out of scope — not a gap, a boundary. They have a fundamentally different construction shape (a connection or DSN, not a namespace/directory pair) and none of the “silent default” problem this package exists to close off: `RedisAdapter/MemcachedAdapter` already force you to pass a real connection, no `null` fallback to guard against. Building one directly from `symfony/cache` isn’t a workaround, it’s simply what those backends need — `$pool->clear()` (part of PSR-6 itself) already works on them the same way it works on everything else.

PhpFilesCache and FilesystemCache

```
use Derafu\Cache\Adapter\PhpFilesCache;
use Derafu\Cache\Adapter\FilesystemCache;

// OPcache-friendly: values must be var_export()-able (arrays/scalars, no objects).
$pool = new PhpFilesCache('my_package', '/var/cache/my_package');
```

```
// Serialized: supports arbitrary values, including objects. No OPcache benefit.  
$pool = new FilesystemCache('my_package', '/var/cache/my_package');
```

Both extend the matching `symfony/cache` adapter directly, under the same name, so anyone already familiar with `symfony/cache` recognizes immediately what each one does. `$namespace` and `$directory` are both required — `$defaultLifetime` (3600 by default) is the only parameter allowed to have one, since it isn't a "where does my data live" decision.

LocalCacheFactory

For the case where the backend itself is a runtime choice — from configuration, the way a per-feature cache: `memory|filesystem` setting would — rather than something you already know at the call site:

```
use Derafu\Cache\LocalCacheFactory;  
use Derafu\Cache\Enum\LocalCacheBackend;  
use Symfony\Component\Cache\Adapter\ArrayAdapter;  
  
$pool = LocalCacheFactory::pool(LocalCacheBackend::Filesystem, 'my_package',  
    '/var/cache/my_package');  
  
// PSR-16 instead of PSR-6 — the same pool, wrapped in Symfony's Psr16Cache bridge.  
$cache = LocalCacheFactory::simple(LocalCacheBackend::Memory, 'my_package');  
  
// Tag-aware — $item->tag([...]) on save, invalidateTags([...]) to drop a whole group  
at once.  
$cache = LocalCacheFactory::taggedPool(LocalCacheBackend::Apcu, 'my_package');  
  
// Several pools, fastest first, with automatic backfill on a slower-tier hit.  
$pool = LocalCacheFactory::layered([new ArrayAdapter(), $filesystemPool]);
```

`$directory` is required for `Filesystem/PhpFiles`, ignored for `None/Memory/Apcu` — passing it for those three is harmless, omitting it for the other two throws.

`LocalCacheBackend::None` (backed by Symfony's own `NullAdapter`) is how "no caching" is expressed — not a runtime flag a consumer has to check, but a pool like any other. A decorator that always receives a `CacheItemPoolInterface` needs no special case for "caching is off": it just does nothing useful when this happens to be the pool it got.

Last updated on 10/10/2026