

Metadata and Exit Codes

Metadata and Exit Codes

Anonymous · 10/10/2026

Metadata and Exit Codes

Execution Metadata: `-v`

The response payload's *shape* never depends on where it is written, only on `-v/--verbose` — and only for how *much* ends up in it, never whether `meta/data_type/timestamp` are there at all (they always are). Without `-v`, `meta` has just `timestamp/data_type`, as shown above. With it, the rest of `ExecutionMetadata` (`startedAt`, timing, memory, CPU, load average) is merged into that same `meta` on success, or into `extensions` on failure (next to the already-present `debug/context/throwable`) — never a new top-level key:

```
bin/console -v billing:invoice:builder:build request.json
```

```
{
  "meta": {
    "timestamp": 1755500000.123456,
    "data_type": "App\\Entity\\Invoice",
    "startedAt": "2026-01-20T10:00:00+00:00",
    "finishedAt": "2026-01-20T10:00:00+00:00",
    "realTime": 0.0234,
    "userTime": 0.0198,
    "systemTime": 0.0012,
    "memoryUsed": 131072,
    "peakMemory": 4194304,
    "pid": 12345,
    "loadAverage1Min": 0.52,
    "loadAverage5Min": 0.61,
    "loadAverage15Min": 0.58
  },
  "data": { "id": "INV-001" }
}
```

Exit Codes

`bin/console help <command>` always lists every exit code that specific command can return — the

fixed ones below, plus whatever the injected `ExitCodeResolverInterface` reports via `describe()` (see next section).

Code	Meaning
0	Success.
1	The operation failed, and nothing more specific applies (<code>DefaultExitCodeResolver</code> 's fallback).
10-16	One of <code>derafu/backbone-dispatcher</code> 's own 7 generic exceptions — see below.
65 (EX_DATAERR)	The request could not be parsed as JSON/YAML/XML.
66 (EX_NOINPUT)	The input file does not exist or is not readable.
70 (EX_SOFTWARE)	An unexpected internal error — a bug, not a usage or business problem.
73 (EX_CANTCREAT)	<code>--output/--error-output</code> could not be created.

65/66/70/73 are `sysexits(3)` codes — a real BSD convention (`/usr/include/sysexits.h` on macOS/BSD), chosen because they start at 64 specifically to avoid colliding with whatever small integers other programs already use for their own exit codes. 2 (Symfony Console's own `Command::INVALID`) is reserved but currently unused by this package.

ExitCodeResolverInterface: Mapping Exceptions to Codes

Every failure not caused by this command itself (a `ProblemDetailInterface`, produced by the dispatch) goes through `ExitCodeResolverInterface::resolve()`. `DefaultExitCodeResolver` — the default — already maps the 7 exceptions generic to *any* Backbone-based project, since they mean the same thing regardless of domain:

Code	Exception
10	<code>OperationNotFoundException</code>
11	<code>OperationNotAllowedException</code>
12	<code>MissingParameterException</code>
13	<code>InvalidParameterTypeException</code>
14	<code>ClassNotFoundException</code>
15	<code>FromArrayMethodNotFoundException</code>
16	<code>NoDeserializerFoundException</code>

A project that also wants its *own* business exceptions mapped extends `DefaultExitCodeResolver` rather than starting from nothing, layering its mapping on top via `parent::`:

```
use Derafu\BackboneConsole\Service\DefaultExitCodeResolver;
use Derafu\BackboneDispatcher\Contract\ProblemDetailInterface;
```

```
class MyExitCodeResolver extends DefaultExitCodeResolver
{
    public function resolve(ProblemDetailInterface $problem): int
    {
        return match ($problem->getThrowable()->getClass()) {
            MyBusinessException::class => 20,
            default => parent::resolve($problem),
        };
    }

    public function describe(): array
    {
        return [MyBusinessException::class => 20] + parent::describe();
    }
}
```

Codes `>= 10` are the convention for a project's own mapping — clear of Symfony Console's `0/1/2`, `DefaultExitCodeResolver`'s own `10-16`, and the `64-78 sysexits(3)` range this package uses internally.

Last updated on 10/10/2026