Query Bundle

Query Bundle

Anonymous · 09/09/2026 · #php, #symfony

Query Bundle

This bundle integrates derafu/query — an expressive path-based query builder for PHP — into a Symfony application by registering its core services in the dependency injection container. If you are not yet familiar with derafu/query, reading its documentation first will give you the necessary context to understand what the bundle exposes.

Requirements

- PHP 8.5 or higher
- Symfony 8.0 or higher (`symfony/framework-bundle`)

Installation

Install the bundle via Composer:

```
composer require derafu/symfony-query-bundle
```

Because the `derafu/query` package is currently tracked on its `dev-main` branch, you may need to set the minimum stability in your `composer.json` to `dev` if it is not already:

```
{
    "minimum-stability": "dev",
    "prefer-stable": true
}
```

Enabling the Bundle

If your application uses Symfony Flex and has bundle auto-discovery enabled, the bundle will be registered automatically. Otherwise, add it manually to `config/bundles.php`:

```
return [
    // ... other bundles
    Derafu\QueryBundle\QueryBundle::class => ['all' => true],
];
```

No other files need to be created or modified for the bundle to function. The services are private by default and are intended to be consumed via autowiring.

Configuration

The bundle works out of the box without any configuration. It automatically resolves the path to the `operators.yaml` file that ships with `derafu/query`.

If you need to use a custom operators file — for example, to add your own operators or override the defaults — create the file `config/packages/derafu_query.yaml` in your application and set the `operators_path` option:

```
derafu_query:
    operators_path: '%kernel.project_dir%/config/query/operators.yaml'
```

When `operators_path` is omitted or set to `null`, the bundle falls back to the operators file bundled with `derafu/query`.

Available Services

All services are registered with autowiring enabled. Inject them by their interface in your controllers, services, or repositories.

Operator system

Interface	Description
`Derafu\Query\Operator\Contract\OperatorLoaderInterface`	Reads operator definitions from a YAML file or array and produces `Operator` objects.
`Derafu\Query\Operator\Contract\OperatorManagerFactoryInterface`	Builds an `OperatorManager` instance from a given operators file path.
`Derafu\Query\Operator\Contract\OperatorManagerInterface`	Provides access to the configured set of operators at runtime.

Parsers

Interface	Description
`Derafu\Query\Filter\Contract\PathParserInterface`	Parses a property path string (e.g. `user.address.city`).
`Derafu\Query\Filter\Contract\ExpressionParserInterface`	Parses a single filter expression (e.g. `name:eq:John`).
`Derafu\Query\Filter\Contract\CompositeExpressionParserInterface`	Parses a composite filter expression combining multiple conditions.
`Derafu\Query\Filter\Contract\FilterParserInterface`	High-level parser that coordinates path and expression parsing.

Doctrine ORM bridge

Interface	Description
`Derafu\Query\Bridge\Contract\QueryBuilderConditionApplierInterface`	Applies a parsed filter condition to a Doctrine ORM `QueryBuilder` instance.

API Platform bridge

The bundle also registers `Derafu\Query\Bridge\ApiPlatform\SmartFilter` and tags it as an `api_platform.filter`. This filter accepts derafu/query expressions as `QueryParameter` values. You can reference it directly in your API Platform resources:

```
use ApiPlatform\Metadata\QueryParameter;
use Derafu\Query\Bridge\ApiPlatform\SmartFilter;

#[QueryParameter(name: 'status', property: 'status', filter: SmartFilter::class)]
```

The `SmartFilter` service is only meaningful if `api-platform/core` is installed. If the package is absent the service definition will still be loaded, but it will not be usable.

Basic Usage

Injecting parsers

```
use Derafu\Query\Filter\Contract\FilterParserInterface;
```

```
final class ProductRepository
{
    public function __construct(
        private readonly FilterParserInterface $filterParser,
    ) {}

    public function findByFilter(string $filterString): array
    {
        $filter = $this->filterParser->parse($filterString);
        // use $filter to build your query
    }
}
```

Applying conditions to a Doctrine ORM QueryBuilder

```
use Derafu\Query\Bridge\Contract\QueryBuilderConditionApplierInterface;
use Derafu\Query\Filter\Contract\FilterParserInterface;

final class ProductRepository extends ServiceEntityRepository
{
    public function __construct(
        ManagerRegistry $registry,
        private readonly FilterParserInterface $filterParser,
        private readonly QueryBuilderConditionApplierInterface $conditionApplier,
    ) {
        parent::__construct($registry, Product::class);
    }

    public function search(string $filterString): array
    {
        $qb = $this->createQueryBuilder('p');
        $filter = $this->filterParser->parse($filterString);
        $this->conditionApplier->apply($qb, $filter);

        return $qb->getQuery()->getResult();
    }
}
```

Refer to the derafu/query documentation for the full expression syntax, the list of available operators, path navigation rules, and security recommendations.

Last updated on 09/09/2026