

Service Configuration

Service Configuration

Anonymous · 30/10/2026

Service Configuration

Shared Operators and Parsers

Import the core configuration to inject Query's operator catalogue and parsers without selecting a database or requiring Doctrine or Illuminate:

```
imports:
  - { resource: '../vendor/derafu/query/resources/config/query-services.yaml' }
```

Each service is registered directly under its interface:

- `OperatorLoaderInterface`, `OperatorManagerFactoryInterface` and `OperatorManagerInterface` in `Derafu\Query\Operator\Contract`.
- `PathParserInterface`, `FilterParserInterface`, `ExpressionParserInterface` and `CompositeExpressionParserInterface` in `Derafu\Query\Filter\Contract`.

The manager is built by the shared factory from the configured operators file.

`derafu.query.operators.path` defaults to the package's `resources/operators.yaml`, located through `OperatorManagerFactory::DEFAULT_OPERATORS_PATH`. No kernel or Composer vendor-directory convention is required to locate that file.

To use a custom catalogue, override the parameter after the imports:

```
parameters:
  derafu.query.operators.path: '%kernel.project_dir%/config/query-operators.yaml'
```

This replaces the catalogue used by the shared parsers; it does not append the file to the built-in definitions. The custom file must contain the types and operators sections expected by `OperatorLoader`.

Optional Integrations

Each optional YAML imports `query-services.yaml` automatically:

File	Services selected	Application supplies
query-engine-pdo-services.yaml	SqlEngineInterface with PdoEngine, plus QueryBuilderInterface with SqlQueryBuilder	A PDO connection service
query-engine-doctrine-services.yaml	SqlEngineInterface with DoctrineEngine, plus QueryBuilderInterface with SqlQueryBuilder	A Doctrine\DBAL\Connection service; requires doctrine/dbal
query-bridge-doctrine-dbal-services.yaml	QueryBuilderConditionApplierInterface with the DBAL implementation	A DBAL query builder at each call; requires doctrine/dbal
query-bridge-doctrine-orm-services.yaml	QueryBuilderConditionApplierInterface with the ORM implementation	An ORM query builder at each call; requires doctrine/orm
query-bridge-illuminate-services.yaml	QueryBuilderConditionApplierInterface with the Illuminate implementation	An Illuminate query builder at each call; requires illuminate/database

The two engine files select alternative implementations of the same engine contract. Likewise, the three bridge files select alternatives for the same applier contract. Importing alternatives together does not create separately addressable implementations: the later registration replaces the earlier one. An engine configuration and a bridge configuration can be combined because they register different contracts.

For example, an application providing a PDO connection can enable SQL execution:

```
imports:
  - { resource: '../vendor/derafu/query/resources/config/query-engine-pdo-services.yaml' }
```

The package does not create connections or choose a DSN, credentials or database. The registered `QueryBuilderInterface` is non-shared: each resolution gets a new mutable builder, while the engine and expression parser remain shared. Use the builder's `table()` or `new()` entry points when starting queries, according to its public contract.

Bridge configurations register only the condition applier and core parsers; the caller supplies its existing query builder for each operation. No connection, entity manager or Eloquent model is created by the bridge configuration.

Application Configuration Loaders and API Platform

`JsonConfigLoader` and `YamlConfigLoader` remain explicit choices. The core YAML does not select a default `ConfigLoaderInterface`.

The API Platform `SmartFilter` is not registered by these files. Its integration belongs to the Symfony bundle or an application's own API Platform composition. See [Framework Bridges](#) for the supported bridge behavior.

Last updated on 30/10/2026