FactoryProvider

August 18, 2026

FactoryProvider

Kind: Interface

Source: `packages/common/interfaces/modules/provider.interface.ts`

Part of: Common

Interface defining a *Factory* type provider.

For example:

```
const connectionFactory = {
  provide: 'CONNECTION',
  useFactory: (optionsProvider: OptionsProvider) => {
    const options = optionsProvider.get();
    return new DatabaseConnection(options);
  },
  inject: [OptionsProvider],
};
```

`FactoryProvider<T>` defines a dependency injection provider that creates a value by invoking a factory function. The factory can receive injected dependencies, return either a synchronous value or a `Promise`, and can be configured with lifecycle scope and durability options.

Properties

Property	Type
`provide`	`InjectionToken`
`useFactory`	`(...args: any[]) => T
`inject`	`Array<InjectionToken
`scope`	`Scope`
`durable`	`boolean`

Diagram

```
graph LR
  Token[Injection Token] --> Provider[FactoryProvider]
  Dependencies[Injected Dependencies] --> Factory[useFactory]
  Provider --> Factory
  Factory --> Instance[Created Value or Promise]
  Instance --> Container[DI Container]
  Provider --> Scope[scope]
  Provider --> Durable[durable]
```

Usage

```
import { Scope } from '@nestjs/common';
import type { FactoryProvider } from '@nestjs/common';

class ConfigService {
  getDatabaseUrl(): string {
    return process.env.DATABASE_URL ?? 'postgres://localhost/app';
  }
}

class DatabaseConnection {
  constructor(public readonly url: string) {}
}

const databaseConnectionProvider: FactoryProvider<DatabaseConnection> = {
  provide: 'DATABASE_CONNECTION',
  inject: [ConfigService],
  scope: Scope.DEFAULT,
  durable: true,
  useFactory: (configService: ConfigService) => {
    return new DatabaseConnection(configService.getDatabaseUrl());
  },
};
```

AI Coding Instructions

- Use `provide` as the token consumers inject; prefer symbols or constants for shared tokens to avoid string collisions.
- List factory parameters in the same order as their corresponding entries in `inject`.
- Return a value directly for synchronous initialization, or return a `Promise` when setup requires asynchronous work.

- Use `OptionalFactoryDependency` in `inject` when a factory dependency is not required, and handle an absent value safely.
- Set `scope` and `durable` intentionally, since they affect provider lifecycle behavior and instance reuse.

Used by

7 references from 7 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (7)

- `LOGGER_PROVIDER` — `integration/scopes/src/resolve-scoped/logger.provider.ts`:3
- `isClassProvider` — `packages/core/injector/helpers/provider-classifier.ts`:9
- `INSTANCE_METADATA_SYMBOL` — `packages/core/injector/instance-wrapper.ts`:22
- `InternalCoreModule` — `packages/core/injector/internal-core-module/internal-core-module.ts`:16
- `Module` — `packages/core/injector/module.ts`:44
- `DependenciesScanner` — `packages/core/scanner.ts`:75
- `superJSONProvider` — `sample/34-using-esm-packages/src/superjson.provider.ts`:7