ClassProvider

August 17, 2026

ClassProvider

Kind: Interface

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

Part of: Common

Interface defining a *Class* type provider.

For example:

```
const configServiceProvider = {
provide: ConfigService,
useClass:
  process.env.NODE_ENV === 'development'
    ? DevelopmentConfigService
    : ProductionConfigService,
};
```

`ClassProvider` defines a dependency injection provider that creates a token's value by instantiating a class. It maps an `InjectionToken` to a concrete implementation type and can optionally configure the provider scope and durability behavior.

Properties

Property	Type
`provide`	`InjectionToken`
`useClass`	`Type<T>`
`scope`	`Scope`
`inject`	`never`
`durable`	`boolean`

Diagram

```
graph LR
  A[InjectionToken<br/>provide] --> B[ClassProvider]
  B --> C[Implementation Class<br/>useClass]
  C --> D[DI Container]
  D --> E[Injected Consumer]

  B --> F[Scope<br/>scope]
  B --> G[Durability<br/>durable]
```

Usage

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

class DevelopmentConfigService {
  getDatabaseUrl() {
    return 'postgres://localhost/dev';
  }
}

class ProductionConfigService {
  getDatabaseUrl() {
    return process.env.DATABASE_URL;
  }
}

const configServiceProvider: ClassProvider = {
  provide: 'CONFIG_SERVICE',
  useClass:
    process.env.NODE_ENV === 'development'
      ? DevelopmentConfigService
      : ProductionConfigService,
  scope: Scope.DEFAULT,
  durable: true,
};

// Register in a module:
// @Module({
//   providers: [configServiceProvider],
//   exports: ['CONFIG_SERVICE'],
// })
```

AI Coding Instructions

- Use `provide` to define the token consumers inject, and `useClass` to define the concrete class instantiated for that token.
- Choose `useClass` when the dependency should be created by the container rather than supplied as an existing value.
- Ensure the class assigned to `useClass` is constructible and that its own constructor dependencies are registered providers.
- Configure `scope` only when lifecycle behavior differs from the default singleton scope.
- Do not add an `inject` array to a `ClassProvider`; constructor dependencies are resolved from the `useClass` type automatically.

Used by

3 references from 3 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (3)

- `isClassProvider` — `packages/core/injector/helpers/provider-classifier.ts`:9
- `Module` — `packages/core/injector/module.ts`:44
- `DependenciesScanner` — `packages/core/scanner.ts`:75