OrphanedEnhancerDefinition

August 18, 2026

OrphanedEnhancerDefinition

Kind: Interface

Source: `packages/core/inspector/interfaces/extras.interface.ts`

Part of: Core

Enhancers registered through "app.useGlobalPipes()", "app.useGlobalGuards()", "app.useGlobalInterceptors()", and "app.useGlobalFilters()" methods.

`OrphanedEnhancerDefinition` describes globally registered NestJS enhancers that are not attached to a specific module provider definition. It is used by the inspector to track pipes, guards, interceptors, and filters registered through application-level methods such as `app.useGlobalGuards()` and preserve both their enhancer subtype and original runtime reference.

Properties

Property	Type
`subtype`	`EnhancerSubtype`
`ref`	`unknown`

Diagram

```
graph LR
  App[Nest Application] -->|useGlobalPipes / Guards / Interceptors / Filters| Enhancer[Global E
  Enhancer --> Definition[OrphanedEnhancerDefinition]
  Definition --> Subtype[subtype: EnhancerSubtype]
  Definition --> Ref[ref: unknown]
  Definition --> Inspector[Core Inspector]
```

Usage

```ts
import type { OrphanedEnhancerDefinition } from '@nestjs/core/inspector/interfaces/extras.inter
import { EnhancerSubtype } from '@nestjs/core/inspector/interfaces/enhancer-metadata-cache.inte

const globalGuard = {
  canActivate: () => true,
};

const definition: OrphanedEnhancerDefinition = {
  subtype: EnhancerSubtype.GUARD,
  ref: globalGuard,
};

// The inspector can retain this definition for later graph analysis.
function registerOrphanedEnhancer(
  enhancer: OrphanedEnhancerDefinition,
): void {
  console.log(`Registered global ${enhancer.subtype}`, enhancer.ref);
}

registerOrphanedEnhancer(definition);
```

AI Coding Instructions

- Use this interface only for enhancers registered globally through `useGlobalPipes`, `useGlobalGuards`, `useGlobalInterceptors`, or `useGlobalFilters`.
- Set `subtype` to the matching `EnhancerSubtype`; do not infer the enhancer kind solely from the shape of `ref`.
- Preserve the original enhancer instance, class, or token in `ref`, since it is intentionally typed as `unknown`.
- Treat these definitions as inspector metadata rather than dependency-injection provider metadata; globally registered enhancers may not have a module ownership relationship.