RouteParamMetadata

August 17, 2026

RouteParamMetadata

Kind: Interface

Source: `packages/common/decorators/http/route-params.decorator.ts`

Part of: Common

`RouteParamMetadata` describes metadata for a parameter in an HTTP route handler. It records the parameter's zero-based position (`index`) and associated route parameter configuration (`data`) so the framework can resolve and inject request values into the correct handler argument.

Properties

Property	Type
`index`	`number`
`data`	`ParamData`

Diagram

```
graph LR
  Decorator[Route parameter decorator] --> Metadata[RouteParamMetadata]
  Metadata --> Index[index: parameter position]
  Metadata --> Data[data: ParamData]
  Metadata --> Resolver[Route argument resolver]
  Resolver --> Handler[Controller handler argument]
```

Usage

```
import type { RouteParamMetadata } from './route-params.decorator';

const userIdParameter: RouteParamMetadata = {
  index: 0,
```

```
  data: 'id',
};

// Used by route metadata processing to inject req.params.id
// into the first controller method argument.
```

AI Coding Instructions

- Keep `index` aligned with the decorated method parameter's zero-based position.
- Use `data` values compatible with the `ParamData` type; do not introduce untyped parameter keys.
- Preserve this metadata shape when extending route parameter decorators, since argument resolvers depend on it.
- Avoid changing property names or semantics without updating metadata readers and request-argument resolution logic.

How it works

`RouteParamMetadata` is an exported TypeScript interface for an entry in HTTP route-argument metadata. Its declared shape has:

- `index: number` — the zero-based route-handler argument position. [packages/common/decorators/http/route-params.decorator.ts:25-28]
- `data?: ParamData` — optional decorator data, where `ParamData` is `object | string | number`. [packages/common/decorators/http/route-params.decorator.ts:24-28]

Standard route-parameter decorators read existing `ROUTE_ARGS_METADATA`, add an entry keyed as `${paramtype}:${index}`, and write the resulting object back through `Reflect.defineMetadata`. The entry records `index`, `data`, and a `pipes` array. [packages/common/decorators/http/route-params.decorator.ts:47-64] The `ROUTE_ARGS_METADATA` metadata key is the string `__routeArguments__`. [packages/common/constants.ts:18]

Although `pipes` is stored and later read, it is not declared in `RouteParamMetadata`. [packages/common/decorators/http/route-params.decorator.ts:25-28] [packages/common/decorators/http/route-params.decorator.ts:37-44] Custom parameter decorators similarly create entries with `index`, `factory`, `data`, and `pipes`; `factory` is also not declared by the interface. [packages/common/utils/assign-custom-metadata.util.ts:9-25]

At HTTP handler setup, the router reads these entries, uses the largest recorded `index` plus one as the argument-array length, and converts each metadata entry into parameter-processing details. [packages/core/router/router-execution-context.ts:191-221] [packages/core/helpers/context-utils.ts:52-56] For standard entries, it passes `data` and the parameter type to `RouteParamsFactory`; for example, `data` selects a body, path, query, host, or header property when it is truthy. [packages/core/router/router-execution-context.ts:315-327] [packages/core/router/route-params-factory.ts:21-44] During invocation, the extracted value is written to `args[index]`; pipeable parameter types run global and parameter pipes first. [packages/core/router/router-execution-context.ts:392-415]

The interface itself has no runtime validation, executable behavior, thrown errors, or side effects; those occur in the decorators and router code that create and consume objects shaped like it. [packages/common/decorators/http/route-params.decorator.ts:25-28]

Used by

1 reference from 1 file. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (1)

- `ParamProperties` — `packages/core/router/router-execution-context.ts`:50