ParseIntPipeOptions

August 17, 2026

ParseIntPipeOptions

Kind: Interface

Source: `packages/common/pipes/parse-int.pipe.ts`

Part of: Common

`ParseIntPipeOptions` configures how NestJS's `ParseIntPipe` validates and transforms incoming values into integers. It lets you customize the HTTP status code or exception factory used for invalid values, and optionally allow `null` or `undefined` values to pass through unchanged.

Properties

Property	Type
`errorHttpStatusCode`	`ErrorHttpStatusCode`
`exceptionFactory`	`(error: string) => any`
`optional`	`boolean`

Diagram

```
graph LR
  A[Incoming request value] --> B[ParseIntPipe]
  B --> C{Value is null/undefined<br/>and optional?}
  C -->|Yes| D[Return original value]
  C -->|No| E{Valid integer?}
  E -->|Yes| F[Return parsed integer]
  E -->|No| G[Create validation exception]
  G --> H[errorHttpStatusCode]
  G --> I[exceptionFactory]
```

Usage

```
import { Controller, Get, Param, ParseIntPipe } from '@nestjs/common';
import type { ParseIntPipeOptions } from '@nestjs/common';

const parseIdOptions: ParseIntPipeOptions = {
  errorHttpStatusCode: 422,
  exceptionFactory: (error) => ({
    statusCode: 422,
    message: error,
    error: 'Validation Error',
  }),
  optional: false,
};

@Controller('users')
export class UsersController {
  @Get(':id')
  findOne(
    @Param('id', new ParseIntPipe(parseIdOptions))
    id: number,
  ) {
    return { id };
  }
}
```

AI Coding Instructions

- Pass `ParseIntPipeOptions` to `new ParseIntPipe(options)` when endpoint-specific integer validation behavior is needed.
- Use `errorHttpStatusCode` for standard HTTP error customization; use `exceptionFactory` when the application requires a custom exception or response shape.
- Set `optional: true` only for parameters that may legitimately be `null` or `undefined`; it does not make invalid non-empty strings valid integers.
- Keep custom exception factories consistent with the application's global error-response conventions.
- Type option objects as `ParseIntPipeOptions` when defining reusable pipe configuration.

How it works

`ParseIntPipeOptions` is the optional configuration interface accepted by `ParseIntPipe`'s constructor. The constructor replaces an omitted options object with `{}`.

packages/common/pipes/parse-int.pipe.ts:17-34 packages/common/pipes/parse-int.pipe.ts:47-50

- `errorHttpStatusCode?: ErrorHttpStatusCode` selects the exception class used when integer validation fails, unless `exceptionFactory` is set. It defaults to `HttpStatus.BAD_REQUEST` (400). packages/common/pipes/parse-int.pipe.ts:21 packages/common/pipes/parse-int.pipe.ts:49-54 packages/common/enums/http-status.enum.ts:26
  `ErrorHttpStatusCode` is limited to the status codes that index `HttpErrorByCode`; that map associates each allowed code with an exception class. packages/common/utils/http-error-by-code.util.ts:27-50
- `exceptionFactory?: (error: string) => any` overrides status-code-based exception creation. On validation failure, the pipe calls it with the exact message `Validation failed (numeric string is expected)` and throws its return value. packages/common/pipes/parse-int.pipe.ts:23-28 packages/common/pipes/parse-int.pipe.ts:52-54 packages/common/pipes/parse-int.pipe.ts:68-72
- `optional?: boolean` defaults to `false`. When it is truthy and the input is `null` or `undefined`, `transform()` returns that input without numeric validation or parsing. packages/common/pipes/parse-int.pipe.ts:30-33 packages/common/pipes/parse-int.pipe.ts:64-67 packages/common/utils/shared.utils.ts:48-49

Without the `optional` nullish branch, the pipe accepts only string or number values whose complete text matches an optional `-` followed by one or more digits and for which `isFinite()` is true; other values trigger the configured exception. Accepted values are returned as `parseInt(value, 10)`. packages/common/pipes/parse-int.pipe.ts:68-85