ParseFloatPipeOptions

August 17, 2026

ParseFloatPipeOptions

Kind: Interface

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

Part of: Common

`ParseFloatPipeOptions` configures the behavior of a float-parsing pipe. It controls whether missing values are accepted, which HTTP status code is used for validation failures, and how parsing errors are converted into exceptions.

Properties

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

Diagram

```
graph LR
  Input[Incoming value] --> Pipe[ParseFloatPipe]
  Pipe --> Optional{optional enabled?}
  Optional -->|Empty value allowed| Result[Return empty value]
  Optional -->|Value required| Parse[Parse numeric float]
  Parse -->|Valid float| Result2[Return parsed number]
  Parse -->|Invalid value| Factory[exceptionFactory]
  Factory --> Error[Throw exception with errorHttpStatusCode]
```

Usage

```ts
import { ParseFloatPipe } from '@nestjs/common';
import type { ParseFloatPipeOptions } from '@nestjs/common';

const options: ParseFloatPipeOptions = {
  optional: false,
  errorHttpStatusCode: 422,
  exceptionFactory: (error: string) => ({
    statusCode: 422,
    message: `Invalid price: ${error}`,
  }),
};

const parsePrice = new ParseFloatPipe(options);

// Example controller usage:
// @Query('price', parsePrice) price: number
```

AI Coding Instructions

- Use `optional: true` only when empty or missing values should bypass float validation.
- Provide an `exceptionFactory` when the default validation exception format does not match the application's error contract.
- Ensure `errorHttpStatusCode` aligns with API validation conventions, such as `400` or `422`.
- Treat successfully transformed values as JavaScript `number` values and validate domain-specific constraints separately.

How it works

`ParseFloatPipeOptions` is the optional constructor-configuration interface for `ParseFloatPipe`. It has three optional fields: `errorHttpStatusCode`, `exceptionFactory`, and `optional`.

- `errorHttpStatusCode` accepts the `ErrorHttpStatusCode` union, which is limited to the HTTP status codes mapped to exception classes in `HttpErrorByCode`. packages/common/utils/http-error-by-code.util.ts:27-48 packages/common/utils/http-error-by-code.util.ts:50-72
- `exceptionFactory` accepts a function receiving an error string and returning the value that `ParseFloatPipe` throws when validation fails. packages/common/pipes/parse-float.pipe.ts:19-24 packages/common/pipes/parse-float.pipe.ts:64-67

- `optional`, when truthy, causes the pipe to return an input that is `null` or `undefined` without numeric validation or parsing. `isNil` identifies exactly `null` and `undefined`. packages/common/pipes/parse-float.pipe.ts:25-29 packages/common/pipes/parse-float.pipe.ts:60-63 packages/common/utils/shared.utils.ts:48-49

When no `exceptionFactory` is configured, `ParseFloatPipe` creates one that constructs the exception class selected by `errorHttpStatusCode`; that status defaults to `HttpStatus.BAD_REQUEST`. packages/common/pipes/parse-float.pipe.ts:43-50 If numeric validation fails, the factory receives the fixed message `Validation failed (numeric string is expected)`. packages/common/pipes/parse-float.pipe.ts:64-67

The associated pipe accepts values whose runtime type is `string` or `number`, whose `parseFloat` result is not `NaN`, and which are finite; valid values are returned as `parseFloat(value)`. packages/common/pipes/parse-float.pipe.ts:60-69 packages/common/pipes/parse-float.pipe.ts:76-81