MulterModuleAsyncOptions

August 17, 2026

MulterModuleAsyncOptions

Kind: Interface

Source: `packages/platform-express/multer/interfaces/files-upload-module.interface.ts`

Part of: Platform Express

`MulterModuleAsyncOptions` configures Multer asynchronously when registering NestJS file upload support. It allows upload options to be provided by an existing provider, a dedicated options factory class, or an inline factory function with dependency injection.

Properties

Property	Type
`useExisting`	`Type<MulterOptionsFactory>`
`useClass`	`Type<MulterOptionsFactory>`
`useFactory`	`( ...args: any[] ) => Promise`
`inject`	`any[]`

Diagram

```
graph LR
  A[MulterModuleAsyncOptions] --> B[useExisting]
  A --> C[useClass]
  A --> D[useFactory]
  A --> E[inject]

  B --> F[MulterOptionsFactory]
  C --> F
  D --> G[MulterModuleOptions]
  E --> D
```

```
  F --> G
  G --> H[MulterModule.registerAsync]
```

Usage

```
import { Module } from '@nestjs/common';
import {
  MulterModule,
  MulterModuleAsyncOptions,
  MulterOptionsFactory,
} from '@nestjs/platform-express';
import { ConfigService } from '@nestjs/config';
import { diskStorage } from 'multer';

const multerOptions: MulterModuleAsyncOptions = {
  inject: [ConfigService],
  useFactory: async (configService: ConfigService) => ({
    storage: diskStorage({
      destination: configService.get<string>('UPLOAD_DIRECTORY', './uploads'),
      filename: (_request, file, callback) => {
        callback(null, `${Date.now()}-${file.originalname}`);
      },
    }),
    limits: {
      fileSize: 5 * 1024 * 1024,
    },
  }),
};

@Module({
  imports: [MulterModule.registerAsync(multerOptions)],
})
export class AppModule {}
```

AI Coding Instructions

- Use exactly one configuration strategy: `useExisting`, `useClass`, or `useFactory`.
- Add required dependencies to `inject` when using `useFactory`; their order must match the factory function parameters.
- Return a valid `MulterModuleOptions` object from the factory, either synchronously or as a `Promise`.
- Prefer `useExisting` when an existing `MulterOptionsFactory` provider should be reused; use `useClass` when Nest should create a dedicated factory provider.

- Configure storage, file-size limits, and file filtering in the returned Multer options rather than directly in controllers.

How it works

`MulterModuleAsyncOptions` is the public TypeScript configuration shape accepted by `MulterModule.registerAsync()`. It extends only the `imports` member of `ModuleMetadata`; its remaining members select how the module resolves `MulterModuleOptions`. `MulterModuleOptions` is an alias for `MulterOptions`. files-upload-module.interface.ts:4 files-upload-module.interface.ts:16-25 multer.module.ts:30

- `imports` is optional through the inherited `ModuleMetadata.imports` member and is copied to the dynamic module's `imports` field. files-upload-module.interface.ts:16-19 multer.module.ts:31-34
- `useFactory` may be a function with arbitrary injected arguments that returns `MulterModuleOptions` directly or as a promise. When present, it becomes the factory for the `MULTER_MODULE_OPTIONS` provider. files-upload-module.interface.ts:22-25 multer.module.ts:63-68
- `inject` supplies that factory's dependency tokens. When `useFactory` is selected and `inject` is absent, the module uses an empty array. files-upload-module.interface.ts:25 multer.module.ts:63-68
- `useExisting` names an existing class-type token for a `MulterOptionsFactory`. The options provider injects that token and calls its `createMulterOptions()` method. That method may return options directly or asynchronously. files-upload-module.interface.ts:9-11 files-upload-module.interface.ts:20 multer.module.ts:70-75
- `useClass` names a class-type `MulterOptionsFactory`. In the absence of `useExisting` and `useFactory`, the module registers the class as a provider, injects it into the options provider, and calls `createMulterOptions()`. files-upload-module.interface.ts:9-11 files-upload-module.interface.ts:21 multer.module.ts:48-57 multer.module.ts:70-75

`useFactory` takes precedence when it is set: the options-provider construction checks it first. A set `useExisting` also prevents registration of `useClass`; the injected factory token then selects `useExisting` before `useClass`. multer.module.ts:48-49 multer.module.ts:63-75

All members declared directly by this interface are optional, and the visible implementation contains no explicit validation or thrown error for a configuration with none of `useFactory`, `useExisting`, or `useClass`. In that case, its class branch constructs

a provider whose token and injection entry are `undefined`. files-upload-module.interface.ts:20-25 multer.module.ts:48-57 multer.module.ts:70-75

Calling `registerAsync` constructs a dynamic module that exports `MULTER_MODULE_OPTIONS` and adds a generated `MULTER_MODULE_ID` value provider. multer.module.ts:30-42