MessagePattern

August 17, 2026

MessagePattern

Kind: Function

Source: `packages/microservices/decorators/message-pattern.decorator.ts`

Part of: Microservices

Subscribes to incoming messages which fulfils chosen pattern.

`MessagePattern()` marks a controller method as a handler for messages matching a specific request pattern. When a configured NestJS microservice transport receives a matching message, it invokes the decorated method and sends its return value back to the requester.

Signature

```
function MessagePattern(metadata: T, transportOrExtras: Transport | symbol | Record<string, any
```

Parameters

Name	Type
`metadata`	`T`
`transportOrExtras`	`Transport
`maybeExtras`	`Record<string, any>`

Returns: `MethodDecorator`

Diagram

```
graph LR
  Client[Microservice Client] -->|send pattern + payload| Transport[Configured Transport]
  Transport --> Server[NestJS Microservice]
```

```
  Server -->|match pattern| Handler["@MessagePattern() handler"]
  Handler -->|response| Transport
  Transport --> Client
```

Usage

```ts
import { Controller } from '@nestjs/common';
import { MessagePattern, Payload } from '@nestjs/microservices';

@Controller()
export class UsersController {
  @MessagePattern({ cmd: 'get_user' })
  async getUser(@Payload() userId: string) {
    return {
      id: userId,
      name: 'Ada Lovelace',
    };
  }
}
```

AI Coding Instructions

- Apply `@MessagePattern()` to methods in controllers registered with a NestJS microservice application.
- Use stable, shared pattern names or objects (for example, `{ cmd: 'get_user' }`) that match the patterns sent by clients.
- Return a value, `Promise`, or `Observable` when the caller expects a request-response result.
- Use `@Payload()` to extract message data and `@Ctx()` when transport-specific context is required.
- Use `@EventPattern()` instead for fire-and-forget events that do not require a response.

Used by

17 references from 17 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (17)

- `ExternalSvcController` — `integration/inspector/src/external-svc/external-svc.controller.ts`:7

- `AppController` — `integration/microservices/src/app.controller.ts`:20
- `KafkaMessagesController` — `integration/microservices/src/kafka/kafka.messages.controller.ts`:9
- `KafkaConcurrentMessagesController` — `integration/microservices/src/kafka-concurrent/kafka-concurrent.messages.controller.ts`:6
- `MqttBroadcastController` — `integration/microservices/src/mqtt/mqtt-broadcast.controller.ts`:11
- `MqttController` — `integration/microservices/src/mqtt/mqtt.controller.ts`:16
- `NatsBroadcastController` — `integration/microservices/src/nats/nats-broadcast.controller.ts`:11
- `NatsController` — `integration/microservices/src/nats/nats.controller.ts`:19

…and 9 more.