MessageHandler

August 18, 2026

MessageHandler

Kind: Interface

Source: `packages/microservices/interfaces/message-handler.interface.ts`

Part of: Microservices

`MessageHandler<TInput, TResult, TContext>` defines the executable portion of a microservice message pipeline. Its `next` method processes incoming data with optional context and resolves either a direct result or an RxJS `Observable`, while metadata fields identify event handlers and carry framework-specific extras.

Properties

Property	Type
`next`	`( data: TInput, ctx?: TContext, ) => Promise<Observable>`
`isEventHandler`	`boolean`
`extras`	`Record<string, any>`

Diagram

```
graph LR
  A[Incoming message data] --> B[MessageHandler.next]
  C[Optional context] --> B
  B --> D{Handler result}
  D --> E[Promise<TResult>]
  D --> F[Promise<Observable<TResult>>]
  G[isEventHandler] --> H[Message routing behavior]
  I[extras] --> J[Additional handler metadata]
```

Usage

```ts
import { Observable, of } from 'rxjs';
import { MessageHandler } from './interfaces/message-handler.interface';

interface CreateUserCommand {
  email: string;
}

interface User {
  id: string;
  email: string;
}

const createUserHandler: MessageHandler<
  CreateUserCommand,
  User,
  { requestId: string }
> = {
  isEventHandler: false,
  extras: {
    transport: 'tcp',
    pattern: 'users.create',
  },

  async next(data, ctx): Promise<Observable<User>> {
    console.log(`Creating user for request ${ctx?.requestId}`);

    const user = {
      id: crypto.randomUUID(),
      email: data.email,
    };

    return of(user);
  },
};

// The framework invokes the handler when a matching message arrives.
const result$ = await createUserHandler.next(
  { email: 'user@example.com' },
  { requestId: 'req-123' },
);
```

AI Coding Instructions

- Implement `next` as an async function that accepts the message payload first and optional transport or request context second.
- Return either `Promise<TResult>` for single direct results or `Promise<Observable<TResult>>` when the response should be streamed.

- Set `isEventHandler` to `true` for fire-and-forget event consumers; use `false` for request-response handlers.
- Store transport-specific or framework-specific metadata in `extras` rather than adding ad hoc handler properties.
- Preserve the generic input, result, and context types so message contracts remain type-safe across the microservice pipeline.

How it works

`MessageHandler<TInput = any, TContext = any, TResult = any>` is a public callable TypeScript interface for a microservice message callback. Its three generic types default to `any`. [message-handler.interface.ts:3-6]

- The handler is invoked with required `data: TInput` and optional `ctx?: TContext`. It must return `Promise<TResult>` or `Promise<Observable<TResult>>`; a direct, non-Promise result is not part of this interface. [message-handler.interface.ts:6-10]
- It may carry an optional `next` handler with the same parameters and return type. [message-handler.interface.ts:11-14]
- It may carry an optional `isEventHandler` boolean and optional `extras` object whose string keys map to `any` values. [message-handler.interface.ts:15-16]

`Server.addHandler()` mutates the callback by assigning its `isEventHandler` and `extras` properties before registration. When an event handler is registered for a pattern that already has a handler, it walks that handler's `next` chain and appends the callback at the tail; otherwise, it replaces or creates the map entry for the normalized pattern. [server.ts:139-158]

For request-scoped event handlers, `ListenersController` checks `handlerRef.next`, invokes it with the original arguments, converts both the current and next results to observables, and returns a `forkJoin` containing both results. Without `next`, it returns the current value unchanged. [listeners-controller.ts:187-202] The request-scoped handler calls this logic only when its `isEventHandler` argument is true. [listeners-controller.ts:273-281]

The interface itself contains no runtime implementation, input validation, error handling, or direct side effects; these members are declarations only. [message-handler.interface.ts:6-17]