TcpSocket

August 18, 2026

TcpSocket

Kind: Class

Source: `packages/microservices/helpers/tcp-socket.ts`

Part of: Microservices

`TcpSocket` wraps a Node.js TCP socket for the microservices transport layer. It manages connection lifecycle events, message transmission, stream data handling, and emission of complete incoming messages after TCP packet framing is processed.

Methods

Method	Signature	Returns
`connect`	`connect(port: number, host: string)`	`void`
`on`	`on(event: string, callback: (err?: any) => void)`	`void`
`once`	`once(event: string, callback: (err?: any) => void)`	`void`
`end`	`end()`	`void`
`sendMessage`	`sendMessage(message: any, callback: (err?: any) => void)`	`void`
`handleSend`	`handleSend(message: any, callback: (err?: any) => void)`	`any`
`handleData`	`handleData(data: Buffer	`string)`
`emitMessage`	`emitMessage(data: string)`	`void`

When something fails

- `TcpSocket` handles failure in 2 places: it logs it and continues in 1, and lets it reach the caller in 1.

Diagram

```
graph LR
  Client[ClientTCP / ServerTCP] --> TcpSocket
  TcpSocket -->|connect / end| NetSocket[Node.js net.Socket]
  TcpSocket -->|sendMessage| Outbound[Framed TCP message]
  Outbound --> NetSocket
  NetSocket -->|data| HandleData[handleData]
  HandleData --> EmitMessage[emitMessage]
  EmitMessage --> MessageListeners[message event listeners]
```

Usage

```
import { Socket } from 'node:net';
import { TcpSocket } from '@nestjs/microservices/helpers/tcp-socket';

const tcpSocket = new TcpSocket(new Socket());

tcpSocket.on('message', (payload: Buffer) => {
  console.log('Received response:', payload.toString());
});

tcpSocket.once('error', (error: Error) => {
  console.error('TCP connection failed:', error);
});

await tcpSocket.connect(3000, '127.0.0.1');

tcpSocket.sendMessage(
  JSON.stringify({
    pattern: 'health.check',
    data: {},
  }),
);

// Close the connection when no further messages are needed.
tcpSocket.end();
```

AI Coding Instructions

- Use `TcpSocket` rather than interacting with the underlying `net.Socket` directly when implementing TCP microservice transport behavior.
- Send protocol-compatible serialized messages; TCP is stream-based, so do not assume one `data` event equals one complete application message.
- Register `message`, `error`, and close-related listeners before connecting or sending messages to avoid missing early socket events.

- Preserve the existing `handleData()` and `emitMessage()` flow when changing framing logic, since it is responsible for reconstructing complete messages from streamed chunks.
- Call `end()` during client or server shutdown to release socket resources cleanly.