TcpClientOptions

August 18, 2026

TcpClientOptions

Kind: Interface

Source: `packages/microservices/interfaces/client-metadata.interface.ts`

Part of: Microservices

`TcpClientOptions` configures a microservice client that communicates over the TCP transport. It defines the target host and port, optional serialization behavior, TLS settings, socket implementation, and the maximum inbound buffer size.

Properties

Property	Type
`transport`	`Transport.TCP`
`options`	`{ host?: string; port?: number; serializer?: Serializer; deserializer?: Deserializer; tlsOptions?: ConnectionOptions; socketClass?: Type<TcpSocket>; maxBufferSize?: number; }`

Diagram

```
graph LR
  Client[TCP Microservice Client] --> Config[TcpClientOptions]
  Config --> Transport[transport: Transport.TCP]
  Config --> Connection[options]
  Connection --> Host[host]
  Connection --> Port[port]
  Connection --> Serialization[serializer / deserializer]
  Connection --> TLS[tlsOptions]
  Connection --> Socket[socketClass]
  Connection --> Buffer[maxBufferSize]
  Client --> Server[TCP Server]
```

Usage

```
import { ClientProxyFactory, Transport } from '@nestjs/microservices';
import type { TcpClientOptions } from '@nestjs/microservices';

const tcpOptions: TcpClientOptions = {
  transport: Transport.TCP,
  options: {
    host: '127.0.0.1',
    port: 3001,
    maxBufferSize: 1024 * 1024,
  },
};

const client = ClientProxyFactory.create(tcpOptions);

client.send({ cmd: 'get_user' }, { id: '123' }).subscribe(response => {
  console.log(response);
});
```

AI Coding Instructions

- Always set `transport` to `Transport.TCP`; this interface is only valid for TCP client configurations.
- Provide `host` and `port` values that match the TCP microservice server configuration.
- Use matching `serializer` and `deserializer` implementations on both client and server when customizing message encoding.
- Configure `tlsOptions` only when connecting to a TLS-enabled TCP server, using compatible Node.js `ConnectionOptions`.
- Set `maxBufferSize` appropriately for expected payload sizes to prevent oversized TCP message buffering.

How it works

`TcpClientOptions` is the public TypeScript configuration interface for a TCP microservice client. Its `transport` field is required and must be `Transport.TCP`; it is one member of the `ClientOptions` union. client-metadata.interface.ts:17-24 client-metadata.interface.ts:34-38

- `options` is optional. When present, it may contain `host`, `port`, `serializer`, `deserializer`, `tlsOptions`, `socketClass`, and `maxBufferSize`. client-metadata.interface.ts:37-51

- `ClientProxyFactory.create()` defaults a missing `options` object to `{}` and, for TCP/default transport selection, constructs `ClientTCP` with those options. client-proxy-factory.ts:42-49 client-proxy-factory.ts:70-73
- `host` and `port` select the connection endpoint. If absent, `ClientTCP` uses `localhost` and `3000`, respectively. client-tcp.ts:30-33 constants.ts:3-4
- `serializer` must implement `serialize(value, options?)`; `deserializer` must implement `deserialize(value, options?)`, which may return a value or promise. If either is absent or falsy, the client initializes an identity serializer or incoming-response deserializer, respectively. serializer.interface.ts:10-12 deserializer.interface.ts:10-15 client-proxy.ts:208-231
- `tlsOptions` has Node TLS `ConnectionOptions` type. When it is set, the client creates its underlying socket with `tls.connect`, merging these options while overriding its `port` and `host`; otherwise it creates a `net.Socket` and explicitly connects it during `connect()`. client-metadata.interface.ts:42-45 client-tcp.ts:65-68 client-tcp.ts:99-121
- `socketClass` is a `TcpSocket` constructor. It defaults to `JsonSocket`. client-metadata.interface.ts:44-45 client-tcp.ts:32-36
- `maxBufferSize` is passed only when `socketClass` is exactly the built-in `JsonSocket`; custom socket classes receive no such constructor option. client-tcp.ts:113-121 For `JsonSocket`, an omitted value defaults to `(512 * 1024 * 1024) / 4` characters. json-socket.ts:7-11 json-socket.ts:21-24
- With `JsonSocket`, received data accumulates in a string buffer. If its length exceeds `maxBufferSize`, the buffer is cleared and `MaxPacketLengthExceededException` is thrown; the base socket catches this, emits an error, and ends the socket. json-socket.ts:30-43 tcp-socket.ts:54-60
- The interface itself declares no runtime validation for option values. The visible TCP constructor reads the fields and initializes serializer/deserializer without checks. client-tcp.ts:30-40