ClientTCP

August 18, 2026

ClientTCP

Kind: Class

Source: `packages/microservices/client/client-tcp.ts`

Part of: Microservices

`ClientTCP` is a TCP-based microservice client that establishes and manages a socket connection to a remote NestJS microservice. It handles connection lifecycle events, routes incoming responses to pending requests, and recreates or closes socket resources when errors or disconnects occur.

Extends: `ClientProxy`

Methods

Method	Signature	Returns
`connect`	`connect()`	`Promise<any>`
`handleResponse`	`handleResponse(buffer: unknown)`	`Promise<void>`
`createSocket`	`createSocket()`	`TcpSocket`
`close`	`close()`	`void`
`registerConnectListener`	`registerConnectListener(socket: TcpSocket)`	`void`
`registerErrorListener`	`registerErrorListener(socket: TcpSocket)`	`void`
`registerCloseListener`	`registerCloseListener(socket: TcpSocket)`	`void`
`handleError`	`handleError(err: any)`	`void`
`handleClose`	`handleClose()`	`void`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`unwrap`	`unwrap()`	`T`

Method	Signature	Returns
`publish`	`publish(partialPacket: ReadPacket, callback: (packet: WritePacket) => any)`	`() => void`
`dispatchEvent`	`dispatchEvent(packet: ReadPacket)`	`Promise<any>`

Properties

Property	Type
`logger`	`any`
`port`	`number`
`host`	`string`
`socketClass`	`Type<TcpSocket>`
`tlsOptions`	`ConnectionOptions`
`maxBufferSize`	`number`
`socket`	`TcpSocket
`connectionPromise`	`Promise
`pendingEventListeners`	`Array<{ event: keyof TcpEvents; callback: TcpEvents[keyof TcpEvents]; }>`

Where it refuses work

- `ClientTCP` stops the work with `Error` when `!this.socket` — “Not initialized. Please call the "connect" method first.”.
- `ClientTCP` stops the work with an early return when `this.connectionPromise`.
- `ClientTCP` stops the work with an early return when `err instanceof EmptyError`.
- `ClientTCP` stops the work with an early return when `!callback`.
- `ClientTCP` stops the work with an early return when `isDisposed || err`.
- `ClientTCP` stops the work with an early return when `this.maxBufferSize !== undefined && this.socketClass === JsonSocket`.

When something fails

- `ClientTCP` handles failure in 1 place: it turns it into a return value in all 1.

Diagram

```
graph LR
  App[Application Service] --> Client[ClientTCP]
  Client --> Connect[connect()]
  Connect --> Socket[createSocket()]
  Socket --> Server[TCP Microservice Server]

  Server --> Response[Incoming Response]
  Response --> HandleResponse[handleResponse()]
  HandleResponse --> Pending[Resolve/Reject Pending Request]

  Socket --> Error[error Event]
  Error --> HandleError[handleError()]
  Socket --> Close[close Event]
  Close --> HandleClose[handleClose()]
```

Usage

```
import { ClientTCP } from '@nestjs/microservices';

async function requestUser(userId: string) {
  const client = new ClientTCP({
    host: '127.0.0.1',
    port: 3001,
  });

  try {
    await client.connect();

    const user = await client
      .send({ cmd: 'get_user' }, { userId })
      .toPromise();

    return user;
  } finally {
    client.close();
  }
}
```

AI Coding Instructions

- Call `connect()` before relying on the client for requests when connection readiness must be guaranteed.
- Use `send()` for request-response messaging and ensure subscriptions/promises are handled so pending requests can be cleaned up.

- Preserve socket lifecycle listener registration when changing `createSocket()` or connection logic; `connect`, `error`, and `close` events are required for reliable recovery.
- Route inbound TCP payloads through `handleResponse()` so response correlation and pending callback resolution continue to work.
- Always call `close()` during application shutdown or when disposing manually created client instances.

How it works

`ClientTCP` is a public `ClientProxy<TcpEvents, TcpStatus>` implementation for a TCP connection. It maintains a wrapped `TcpSocket`, a cached in-progress connection promise, pending socket-event listeners, and a request-ID-to-callback routing map inherited from `ClientProxy`. client-tcp.ts:16-28 client-proxy.ts:38-45

Construction and configuration

The constructor accepts TCP client options, selecting port `3000`, host `localhost`, and `JsonSocket` when those options are absent; it also reads optional TLS settings, a custom socket class, and `maxBufferSize`. client-tcp.ts:30-40 constants.ts:3-4 client-metadata.interface.ts:37-51

It initializes the serializer from `options.serializer` or defaults to `IdentitySerializer`, whose `serialize()` returns its input unchanged. It initializes the deserializer from `options.deserializer` or defaults to `IncomingResponseDeserializer`. client-proxy.ts:208-231 identity.serializer.ts:3-6

Without a custom socket class, `JsonSocket` writes packets as `<JSON-string-length>#<JSON>`, parses that framing from received data, and emits each parsed JSON value as a `message` event. json-socket.ts:26-28 json-socket.ts:30-89 json-socket.ts:92-97

`maxBufferSize` is passed only when the selected socket class is exactly `JsonSocket`; custom socket classes receive only the underlying socket. client-tcp.ts:113-121 `JsonSocket` defaults its buffer limit to `(512 * 1024 * 1024) / 4` characters and throws `MaxPacketLengthExceededException` after clearing its buffer when the accumulated buffer exceeds that limit. json-socket.ts:7-24 json-socket.ts:39-43

Connecting and status

`connect()` returns the existing connection promise if one is already stored. Otherwise, it creates a socket, registers connect, close, and error listeners, attaches listeners registered before connection, and clears the pending-listener list. client-tcp.ts:42-54

For non-TLS configuration, it calls the wrapped socket's `connect(port, host)` method. With TLS options, `createSocket()` calls `tlsConnect()` with the TLS options plus the selected host and port; therefore `connect()` does not separately call `socket.connect()` in that case. client-tcp.ts:65-68 client-tcp.ts:99-121

The returned promise resolves or rejects from the first underlying `connect` or `error` event. An RxJS `EmptyError` is converted into a resolved `undefined` result; other errors are rethrown. After the connection event, it subscribes to wrapped-socket `message` events and forwards their payloads to `handleResponse()`. client-tcp.ts:56-76 client-proxy.ts:165-177

On `connect`, the client emits `TcpStatus.CONNECTED` through its inherited status stream. On `close`, it emits `TcpStatus.DISCONNECTED` and runs close handling. On an `error` whose `code` is `ECONNREFUSED`, it emits `DISCONNECTED`; other errors are passed to `handleError()`, which logs them through Nest's `Logger`. client-tcp.ts:129-154 The inherited `status` getter exposes that stream and suppresses consecutive duplicate statuses. client-proxy.ts:47-52

Requests, events, and responses

Inherited `send(pattern, data)` rejects a `null` or `undefined` pattern or data with `InvalidMessageException`. Otherwise, on subscription, it connects and invokes this class's `publish()` method. client-proxy.ts:86-101

`publish()` assigns a generated packet ID, serializes the packet, stores the response callback under that ID, and sends the serialized packet through the current socket. It returns an unsubscription function that removes that callback from the routing map. If ID assignment, serialization, map insertion, or sending throws synchronously, it calls the callback with `{ err }` and returns a no-op cleanup function. client-tcp.ts:189-205 client-proxy.ts:160-163

Inherited `emit(pattern, data)` has the same `null` / `undefined` validation, connects, and calls `dispatchEvent()`. client-proxy.ts:111-126 `dispatchEvent()` normalizes the pattern to a route, serializes the resulting packet, and sends it through the current socket. client-tcp.ts:207-214 client-proxy.ts:204-205

For each received `message`, `handleResponse()` deserializes the payload, finds the callback by deserialized ID, and does nothing if no callback exists. When the response

has `isDisposed` or `err`, it invokes the callback with `isDisposed: true`; otherwise it invokes it with the error and response values as deserialized. client-tcp.ts:79-97

The default response deserializer treats packets already containing `err`, `response`, or `isDisposed` as response-shaped values. Other values are mapped to `{ id, response: value, isDisposed: true }`. incoming-response.deserializer.ts:7-32

Socket access and lifecycle

`on(event, callback)` attaches the listener immediately when a socket exists; before connection, it queues the listener for attachment during `connect()`. Its typed event set includes `error`, `connect`, `end`, `close`, `timeout`, `drain`, and `lookup`. client-tcp.ts:169-178 tcp.events.ts:26-39

`unwrap<T>()` returns the underlying Node socket. It throws `Error('Not initialized. Please call the "connect" method first.')` when no wrapped socket currently exists. client-tcp.ts:180-187

`close()` ends the current wrapped socket when present, runs close handling, and discards queued event listeners. client-tcp.ts:123-127 Close handling clears the socket and cached connection promise. If request callbacks remain, it calls each with `Error('Connection closed')` and clears the routing map. client-tcp.ts:156-167

Relationships

- IMPORTS → `Logger`
- IMPORTS → `Type`

Used by

2 references from 2 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Imported by (2)

- `ConfigService` — `integration/microservices/src/app.module.ts`:18
- `ConfigService` — `integration/microservices/src/tcp-tls/app.module.ts`:29