ClientProxy

August 18, 2026

ClientProxy

Kind: Class

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

Part of: Microservices

`ClientProxy` is the base abstraction for communicating with NestJS microservices through a configured transport. It manages connection lifecycle, request-response messaging via `send()`, and event-based messaging via `emit()`, while delegating transport-specific publishing and packet handling to implementations.

Methods

Method	Signature	Returns
`connect`	`connect()`	`Promise<any>`
`close`	`close()`	`any`
`on`	`on(event: EventKey, callback: EventCallback)`	`void`
`unwrap`	`unwrap()`	`T`
`send`	`send(pattern: any, data: TInput)`	`Observable<TResult>`
`emit`	`emit(pattern: any, data: TInput)`	`Observable<TResult>`
`publish`	`publish(packet: ReadPacket, callback: (packet: WritePacket) => void)`	`() => void`
`dispatchEvent`	`dispatchEvent(packet: ReadPacket)`	`Promise<T>`
`createObserver`	`createObserver(observer: Observer<T>)`	`(packet: WritePacket) => void`
`serializeError`	`serializeError(err: any)`	`any`
`serializeResponse`	`serializeResponse(response: any)`	`any`

Method	Signature	Returns
`assignPacketId`	`assignPacketId(packet: ReadPacket)`	`ReadPacket & PacketId`
`connect$`	`connect$(instance: any, errorEvent: undefined, connectEvent: undefined)`	`Observable<any>`
`getOptionsProp`	`getOptionsProp(obj: Options, prop: Attribute)`	`Options[Attribute]`
`getOptionsProp`	`getOptionsProp(obj: Options, prop: Attribute, defaultValue: DefaultValue)`	`Required<Options>[Attribute]`
`getOptionsProp`	`getOptionsProp(obj: Options, prop: Attribute, defaultValue: DefaultValue)`	`void`
`normalizePattern`	`normalizePattern(pattern: MsPattern)`	`string`
`initializeSerializer`	`initializeSerializer(options: ClientOptions['options'])`	`void`
`initializeDeserializer`	`initializeDeserializer(options: ClientOptions['options'])`	`void`

Properties

Property	Type
`routingMap`	`any`
`serializer`	`ProducerSerializer`
`deserializer`	`ProducerDeserializer`
`_status$`	`any`

Where it refuses work

- `ClientProxy` stops the work with an early return when `isNil(pattern) || isNil(data)`, in 2 places.
- `ClientProxy` stops the work with an early return when `isDisposed`.

Diagram

```
graph LR
  App[Application Service] --> ClientProxy
  ClientProxy --> Connect[connect()]
  ClientProxy --> Send[send() request-response]
  ClientProxy --> Emit[emit() event]
  Send --> Publish[publish()]
  Emit --> Dispatch[dispatchEvent()]
  Publish --> Transport[Configured Transport]
  Dispatch --> Transport
  Transport --> Observer[createObserver()]
  Observer --> App
```

Usage

```
import { ClientProxy, ClientProxyFactory, Transport } from '@nestjs/microservices';
import { firstValueFrom } from 'rxjs';

const client: ClientProxy = ClientProxyFactory.create({
  transport: Transport.TCP,
  options: {
    host: 'localhost',
    port: 3001,
  },
});

async function getUser(userId: string) {
  await client.connect();

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

  return user;
}

async function notifyUserCreated(user: { id: string; email: string }) {
  await firstValueFrom(
    client.emit('user.created', user),
  );
}

async function shutdown() {
  client.close();
}
```

AI Coding Instructions

- Use `send()` for request-response patterns and subscribe to or convert its returned `Observable` with `firstValueFrom()`.
- Use `emit()` for fire-and-forget events; consumers should register matching event patterns.
- Ensure the proxy is connected before sending messages when using manually created clients, and close it during application shutdown.
- Keep transport-specific behavior inside concrete `ClientProxy` implementations; use `publish()` and `dispatchEvent()` as extension points rather than duplicating serialization logic.
- Preserve error serialization through `serializeError()` so remote exceptions remain consistent across transports.

Relationships

- IMPORTS → `randomStringGenerator`
- IMPORTS → `isNil`

Used by

20 references from 13 files. Each is a place in this repository where the symbol is actually used — go read one rather than trusting an example.

Injected or called by (7)

- `AppController` — `integration/microservices/src/app.controller.ts`:20
- `AppController` — `integration/microservices/src/app.controller.ts`:20
- `AppController` — `integration/microservices/src/app.controller.ts`:20
- `AppController` — `integration/microservices/src/tcp-tls/app.controller.ts`:22
- `AppController` — `integration/microservices/src/tcp-tls/app.controller.ts`:22
- `AppController` — `integration/microservices/src/tcp-tls/app.controller.ts`:22
- `MathController` — `sample/03-microservices/src/math/math.controller.ts`:6

Imported by (13)

- `AppController` — `integration/microservices/src/app.controller.ts`:20
- `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
- `RedisBroadcastController` — `integration/microservices/src/redis/redis-broadcast.controller.ts`:11
- `RedisController` — `integration/microservices/src/redis/redis.controller.ts`:12
- `RMQFanoutExchangeProducerController` — `integration/microservices/src/rmq/fanout-exchange-producer-rmq.controller.ts`:9

…and 5 more.