



MICROSERVICES

ClientRMQ

Kind: Class

VERSION
47

UPDATED
7 August 2026

DOCUMENT
reference-subsystem-packages-microservices/class-clientrmq

CONTENTS

1	Methods	2	5	Diagram	4
2	Properties	3	6	AI Coding Instructions	5
3	Where it refuses work	4	7	How it works	5
4	When something fails	4	8	Relationships	5

Kind: Class

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

Part of: [Microservices](#)

`ClientRMQ` is the RabbitMQ transport client used by NestJS microservices to establish AMQP connections, create channels, and send or emit messages. It manages connection lifecycle concerns such as channel setup, error handling, disconnect events, and reconnection through `AmqpConnectionManager`.

Extends: `ClientProxy`

1 Methods

Method	Signature	Returns
<code>close</code>	<code>close()</code>	<code>Promise<void></code>
<code>connect</code>	<code>connect()</code>	<code>Promise<any></code>
<code>createChannel</code>	<code>createChannel()</code>	<code>Promise<void></code>
<code>createClient</code>	<code>createClient()</code>	<code>AmqpConnectionManager</code>
<code>mergeDisconnectEvent</code>	<code>mergeDisconnectEvent(instance: any, source\$: Observable<T>)</code>	<code>Observable<T></code>
<code>convertConnectionToPromise</code>	<code>convertConnectionToPromise()</code>	<code>void</code>
<code>setupChannel</code>	<code>setupChannel(channel: Channel, resolve: Function)</code>	<code>void</code>
<code>consumeChannel</code>	<code>consumeChannel(channel: Channel)</code>	<code>void</code>
<code>registerErrorListener</code>	<code>registerErrorListener(client: AmqpConnectionManager)</code>	<code>void</code>
<code>registerDisconnectListener</code>	<code>registerDisconnectListener(client: AmqpConnectionManager)</code>	<code>void</code>
<code>registerBlockedListener</code>	<code>registerBlockedListener(client: AmqpConnectionManager)</code>	<code>void</code>

registerUnblockedListener	registerUnblockedListener(client: AmqpConnectionManager)	void
on	on(event: EventKey, callback: EventCallback)	void
unwrap	unwrap()	T
handleMessage	handleMessage(packet: unknown, callback: (packet: WritePacket) => any)	Promise<void>
handleMessage	handleMessage(packet: unknown, options: Record<string, unknown>, callback: (packet: WritePacket) => any)	Promise<void>
handleMessage	handleMessage(packet: unknown, options:	Record<string, unknown>
publish	publish(message: ReadPacket, callback: (packet: WritePacket) => any)	() => void
dispatchEvent	dispatchEvent(packet: ReadPacket)	Promise<any>
initializeSerializer	initializeSerializer(options: RmqOptions['options'])	void
mergeHeaders	mergeHeaders(requestHeaders: Record<string, string>)	Record<string, string>
parseMessageContent	parseMessageContent(content: Buffer)	void

2 Properties

Property	Type
logger	any
connection\$	ReplaySubject<any>
connectionPromise	Promise<void>
client	AmqpConnectionManager
channel	ChannelWrapper

<code>pendingEventListeners</code>	<code>Array<{ event: keyof RmqEvents; callback: RmqEvents[keyof RmqEvents]; }></code>
<code>isInitialConnect</code>	<code>any</code>
<code>responseEmitter</code>	<code>EventEmitter</code>
<code>queue</code>	<code>string</code>
<code>queueOptions</code>	<code>Record<string, any></code>
<code>replyQueue</code>	<code>string</code>
<code>noAssert</code>	<code>boolean</code>

3 Where it refuses work

- `ClientRMQ` stops the work with `Error` when `!this.client` — “Not initialized. Please call the "connect" method first.”.
- `ClientRMQ` stops the work with an early return when `this.client`.
- `ClientRMQ` stops the work with an early return when `urls.indexOf(error.url) >= urls.length - 1`.
- `ClientRMQ` stops the work with an early return when `err instanceof EmptyError`.
- `ClientRMQ` stops the work with an early return when `isDisposed || err`.
- `ClientRMQ` stops the work with an early return when `!requestHeaders && !this.options?.headers`.

4 When something fails

- `ClientRMQ` handles failure in 3 places: it turns it into a return value in 2, and lets it reach the caller in 1.

5 Diagram

```
graph LR
  A[Application Service] --> B[ClientRMQ]
  B --> C[createClient()]
  C --> D[AmqpConnectionManager]
  D --> E[RabbitMQ Broker]
  B --> F[createChannel()]
  F --> G[AMQP Channel]
  B --> H[registerErrorListener()]
```

```
I --> J[mergeDisconnectEvent()]\nJ --> K[Reconnect / Channel Setup]
```

6 AI Coding Instructions

- Call `connect()` before publishing messages so the AMQP connection and channel are initialized.
- Use `emit()` for event-based messaging and `send()` when a request/response interaction is required.
- Preserve the existing disconnect and error listener flow; these listeners support channel recovery after RabbitMQ connection failures.
- Configure durable queues and matching queue options consistently between clients and consumers to avoid broker-side queue declaration errors.
- Always call `close()` during application shutdown or test cleanup to release RabbitMQ connections and channels.

7 How it works

`ClientRMQ` is the RabbitMQ client implementation of `ClientProxy`, parameterized with RabbitMQ connection events and statuses. It manages an `amqp-connection-manager` client and channel wrapper, publishes request/reply messages and events, and exposes RabbitMQ connection status through the inherited `status` observable. [packages/microservices/client/client-rmq.ts:57-72](#)
[packages/microservices/client/client-proxy.ts:45-52](#)

8 Relationships

- IMPORTS → `Logger`
- IMPORTS → `loadPackage`
- IMPORTS → `randomStringGenerator`
- IMPORTS → `isFunction`
- IMPORTS → `isString`