ClientRMQ

August 17, 2026

ClientRMQ

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`

Methods

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

Method	Signature	Returns
`registerDisconnectListener`	`registerDisconnectListener(client: AmqpConnectionManager)`	`void`
`registerBlockedListener`	`registerBlockedListener(client: AmqpConnectionManager)`	`void`
`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`

Properties

Property	Type
`logger`	`any`
`connection$`	`ReplaySubject<any>`

Property	Type
`connectionPromise`	`Promise<void>`
`client`	`AmqpConnectionManager
`channel`	`ChannelWrapper
`pendingEventListeners`	`Array<{ event: keyof RmqEvents; callback: RmqEvents[keyof RmqEvents]; }>`
`isInitialConnect`	`any`
`responseEmitter`	`EventEmitter`
`queue`	`string`
`queueOptions`	`Record<string, any>`
`replyQueue`	`string`
`noAssert`	`boolean`

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`.

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.

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()]
B --> I[registerDisconnectListener()]
I --> J[mergeDisconnectEvent()]
J --> K[Reconnect / Channel Setup]
```

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.

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

Relationships

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