WsAdapter

August 18, 2026

WsAdapter

Kind: Class

Source: `packages/platform-ws/adapters/ws-adapter.ts`

Part of: Platform Ws

`WsAdapter` is Nest's WebSocket adapter for the `ws` library. It creates and manages WebSocket servers, routes incoming messages to gateway handlers, and coordinates connection errors, client disconnects, and server cleanup.

Extends: `AbstractWsAdapter`

Methods

Method	Signature	Returns
`create`	`create(port: number, options: Record<string, any> & { namespace?: string; server?: any; path?: string; })`	`void`
`bindMessageHandlers`	`bindMessageHandlers(client: any, handlers: MessageMappingProperties[], transform: (data: any) => Observable<any>)`	`void`
`bindMessageHandler`	`bindMessageHandler(buffer: any, handlersMap: Map<string, MessageMappingProperties>, transform: (data: any) => Observable<any>)`	`Observable<any>`
`bindErrorHandler`	`bindErrorHandler(server: any)`	`void`
`bindClientDisconnect`	`bindClientDisconnect(client: any, callback: Function)`	`void`
`close`	`close(server: any)`	`void`
`dispose`	`dispose()`	`void`

Method	Signature	Returns
`setMessageParser`	`setMessageParser(parser: WsMessageParser)`	`void`
`ensureHttpServerExists`	`ensureHttpServerExists(port: number, httpServer: undefined)`	`void`
`addWsServerToRegistry`	`addWsServerToRegistry(wsServer: T, port: number, path: string)`	`void`

Properties

Property	Type
`logger`	`any`
`httpServersRegistry`	`any`
`wsServersRegistry`	`any`
`messageParser`	`WsMessageParser`

Where it refuses work

- `WsAdapter` stops the work with an early return when `server`.
- `WsAdapter` stops the work with an early return when `client.readyState !== READY_STATE.OPEN_STATE`.
- `WsAdapter` stops the work with an early return when `!message`.
- `WsAdapter` stops the work with an early return when `this.httpServersRegistry.has(port)`.

When something fails

- `WsAdapter` handles failure in 2 places: it logs it and continues in 1, and turns it into a return value in 1.

Diagram

```
graph LR
  App[Nest Application] --> Adapter[WsAdapter]
  Adapter --> Http[HTTP Server]
  Adapter --> WsServer[ws.Server]
  Client[WebSocket Client] --> WsServer
  WsServer --> Parser[Message Parser]
  Parser --> Handler[Gateway Message Handler]
```

```
  Handler --> Response[Observable Response]
  Response --> Client
  WsServer --> Errors[Error Handler]
  Client --> Disconnect[Disconnect Handler]
```

Usage

```
import { NestFactory } from '@nestjs/core';
import { WsAdapter } from '@nestjs/platform-ws';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  // Attach the ws-based adapter to Nest's underlying HTTP server.
  app.useWebSocketAdapter(new WsAdapter(app.getHttpServer()));

  await app.listen(3000);
}

bootstrap();
```

AI Coding Instructions

- Use `WsAdapter` when the application should use the `ws` package instead of the default Socket.IO WebSocket transport.
- Pass the Nest HTTP server to the adapter so WebSocket connections can share the application's existing server and port.
- Keep message parsing compatible with gateway handler payloads; use `setMessageParser()` only when clients send a custom message format.
- Ensure message handlers return values or `Observable`s that can be serialized and sent back to connected clients.
- Call application shutdown hooks or adapter cleanup paths so registered WebSocket servers are closed through `close()` or `dispose()`.

How it works

`WsAdapter` is a public WebSocket adapter class in `@nestjs/platform-ws`. It extends `AbstractWsAdapter` and loads the `ws` package during construction. [packages/platform-ws/adapters/ws-adapter.ts:36-63] The parent adapter stores either the underlying HTTP server from a `NestApplication` argument or the supplied object directly. [packages/websockets/adapters/ws-adapter.ts:29-35]

The constructor accepts an optional application/HTTP-server object and an optional `{ messageParser }` option. Its default parser converts incoming data to a string and runs `JSON.parse`; a supplied parser replaces that default. [packages/platform-ws/adapters/ws-adapter.ts:49-63]

Server creation

`create(port, options)` creates or returns a WebSocket server according to `port`, `server`, and `path`. [packages/platform-ws/adapters/ws-adapter.ts:65-126]

- A truthy `options.namespace` is unsupported: the adapter logs an `Error` and throws it. [packages/platform-ws/adapters/ws-adapter.ts:78-84]
- When called with port `0` and the adapter has an inherited HTTP server, it registers that HTTP server, creates a `ws.Server` with `noServer: true`, registers the WebSocket server under `path`, and returns it. [packages/platform-ws/adapters/ws-adapter.ts:86-96]
- When `options.server` is set outside that branch, it returns that object without creating or registering another server. [packages/platform-ws/adapters/ws-adapter.ts:99-101]
- When `path` is set and `port` is not `0`, it creates or reuses an internal Node HTTP server for the port, calls `listen(port)`, creates a `noServer` `ws.Server`, registers it by path, and returns it. [packages/platform-ws/adapters/ws-adapter.ts:102-116]
- Otherwise, it creates and returns a `ws.Server` with `port`, `path`, and remaining options. [packages/platform-ws/adapters/ws-adapter.ts:118-125]

For registered `noServer` instances, the internal HTTP server handles `upgrade` events. It parses the request pathname, finds the first registered WebSocket server whose normalized `path` exactly matches, then calls `handleUpgrade` and emits that server's `connection` event. [packages/platform-ws/adapters/ws-adapter.ts:217-231] An unmatched pathname destroys the socket; an exception while handling the upgrade writes an HTTP `400` response with the exception message. [packages/platform-ws/adapters/ws-adapter.ts:233-238] Registration normalizes the path and stores servers in a per-port array. [packages/platform-ws/adapters/ws-adapter.ts:243-253]

Incoming messages and responses

`bindMessageHandlers(client, handlers, transform)` indexes the supplied handler metadata by each handler's `message` value, then subscribes to the client's `'message'` events until its first close event. [packages/platform-ws/adapters/ws-adapter.ts:128-144] Each

incoming event is passed to `bindMessageHandler`; `null` and `undefined` results are filtered out. [packages/platform-ws/adapters/ws-adapter.ts:137-143]

`bindMessageHandler` calls the configured parser with `buffer.data`. The parser must return an object with `event` and `data` properties for dispatch to continue. [packages/platform-ws/adapters/ws-adapter.ts:154-166] It finds the handler mapped to `event`, calls that handler's callback with `(data, event)`, and passes the callback result to `transform`. [packages/platform-ws/adapters/ws-adapter.ts:164-166] If parsing returns no message, or if an exception occurs during this sequence, it returns the empty RxJS observable, so no response is emitted from that message. [packages/platform-ws/adapters/ws-adapter.ts:159-169]

For every non-nil value emitted after transformation, the adapter sends `JSON.stringify(response)` to the client only when `client.readyState` equals `1` (`OPEN_STATE`). [packages/platform-ws/adapters/ws-adapter.ts:145-151] Handler metadata declares callbacks that may return an `Observable` or `Promise`. [packages/websockets/gateway-metadata-explorer.ts:15-20]

`setMessageParser(parser)` replaces the parser used by later calls to `bindMessageHandler`. [packages/platform-ws/adapters/ws-adapter.ts:204-206]

Connection, error, and shutdown behavior

`bindErrorHandler(server)` attaches error listeners to both the server and each connected WebSocket client; both listeners log the received error. [packages/platform-ws/adapters/ws-adapter.ts:172-178] `bindClientDisconnect(client, callback)` attaches `callback` to the client's close event. [packages/platform-ws/adapters/ws-adapter.ts:180-182]

`close(server)` calls `server.close`, terminates every entry in `server.clients`, then resolves when the close callback has no error or rejects with that callback error. [packages/platform-ws/adapters/ws-adapter.ts:184-192]

`dispose()` closes internally registered HTTP servers except the server registered under port `0`, waits for all close callbacks, and clears both HTTP-server and WebSocket-server registries. [packages/platform-ws/adapters/ws-adapter.ts:194-202]

Relationships

- IMPORTS → `INestApplicationContext`
- IMPORTS → `Logger`

- IMPORTS → `loadPackage`
- IMPORTS → `isNil`
- IMPORTS → `normalizePath`
- IMPORTS → `AbstractWsAdapter`
- IMPORTS → `CLOSE_EVENT`
- IMPORTS → `CONNECTION_EVENT`
- IMPORTS → `ERROR_EVENT`
- IMPORTS → `MessageMappingProperties`