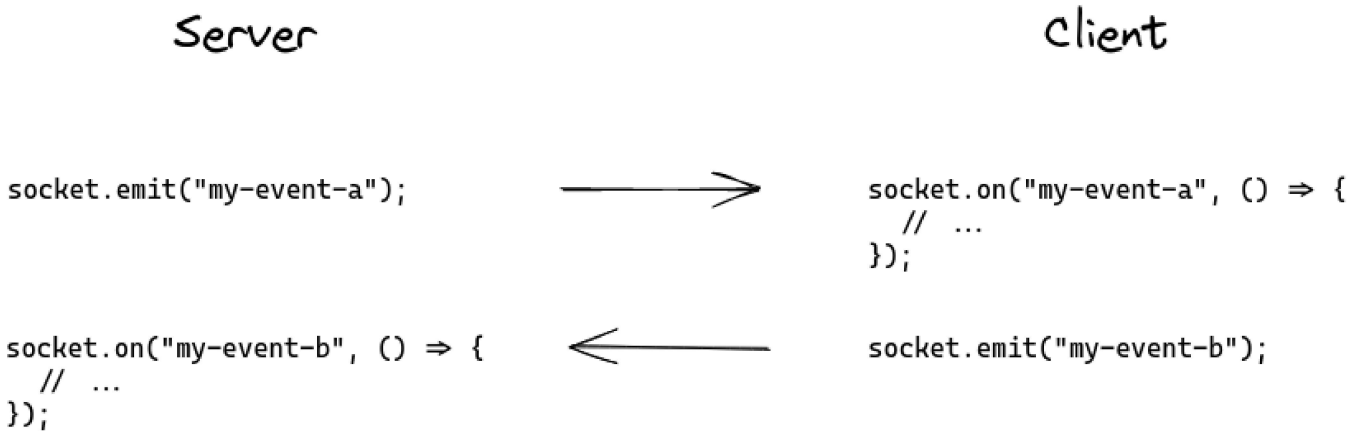
Version: 4.x

On this page

The Socket instance (client-side)

A `Socket` is the fundamental class for interacting with the server. It inherits most of the methods of the Node.js EventEmitter, like emit, on, once or off.



Besides emitting and listening to events, the Socket instance has a few attributes that may be of use in your application:

Socket#id

Each new connection is assigned a random 20-characters identifier.

This identifier is synced with the value on the server-side.

```
// server-side
io.on("connection", (socket) => {
  console.log(socket.id); // x8WIv7-mJelg7on_ALbx
});

// client-side
socket.on("connect", () => {
  console.log(socket.id); // x8WIv7-mJelg7on_ALbx
```

```
});

socket.on("disconnect", () => {
  console.log(socket.id); // undefined
});
```

⚠ CAUTION

The `id` attribute is an **ephemeral** ID that is not meant to be used in your application (or only for debugging purposes) because:

- this ID is regenerated after each reconnection (for example when the WebSocket connection is severed, or when the user refreshes the page)
- two different browser tabs will have two different IDs
- there is no message queue stored for a given ID on the server (i.e. if the client is disconnected, the messages sent from the server to this ID are lost)

Please use a regular session ID instead (either sent in a cookie, or stored in the localStorage and sent in the `auth` payload).

See also:

- Part II of our private message guide
- How to deal with cookies

Socket#connected

This attribute describes whether the socket is currently connected to the server.

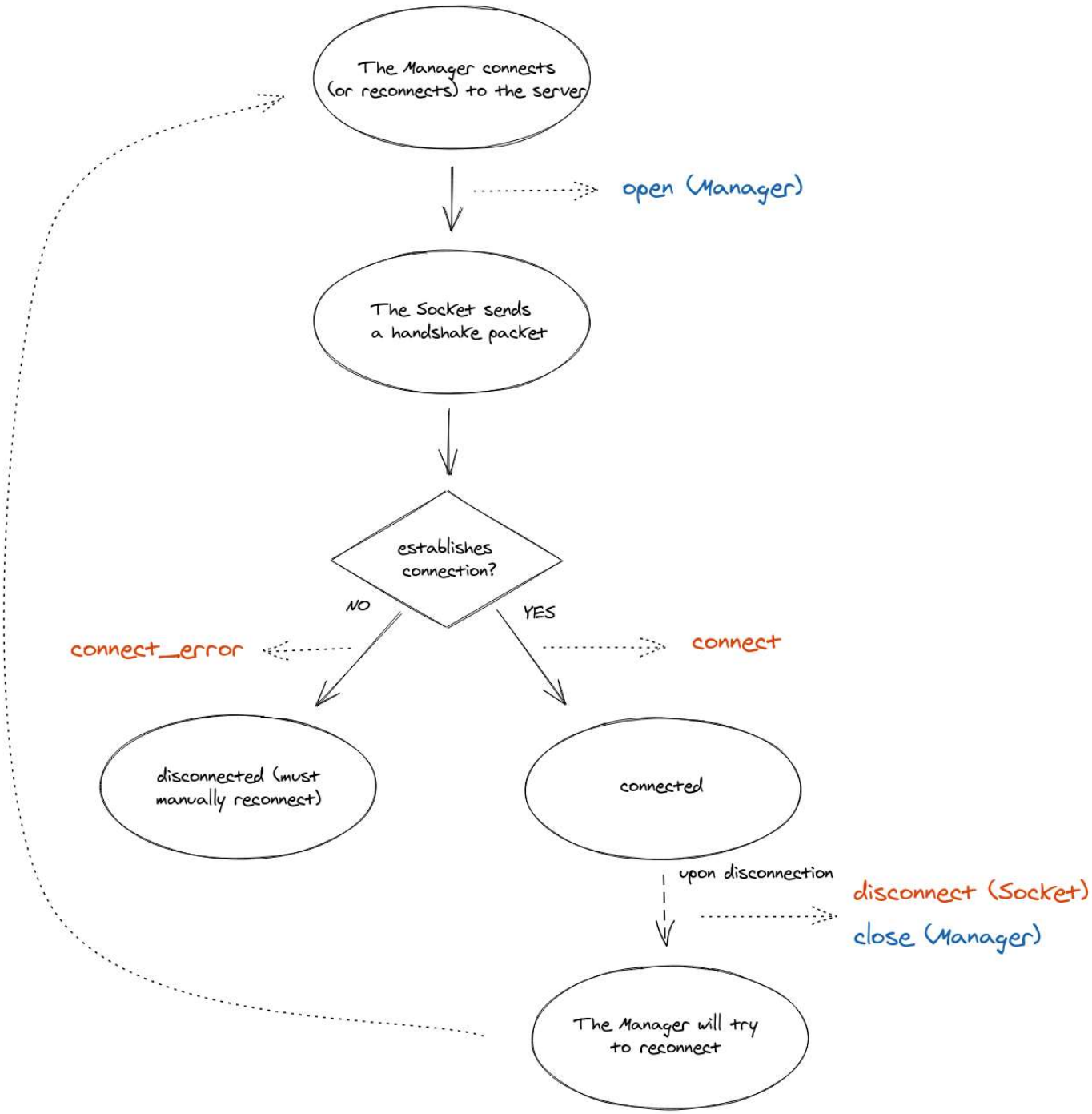
```
socket.on("connect", () => {
  console.log(socket.connected); // true
});

socket.on("disconnect", () => {
  console.log(socket.connected); // false
});
```

Socket#io

A reference to the underlying Manager.

```
socket.on("connect", () => {
  const engine = socket.io.engine;
  console.log(engine.transport.name); // in most cases, prints "polling"

  engine.once("upgrade", () => {
    // called when the transport is upgraded (i.e. from HTTP long-polling to
WebSocket)
    console.log(engine.transport.name); // in most cases, prints "websocket"
  });

  engine.on("packet", ({ type, data }) => {
    // called for each packet received
  });

  engine.on("packetCreate", ({ type, data }) => {
    // called for each packet sent
  });

  engine.on("drain", () => {
    // called when the write buffer is drained
  });

  engine.on("close", (reason) => {
    // called when the underlying connection is closed
  });
});
```

Lifecycle



Events

The Socket instance emits three special events:

- `connect`
- `connect_error`
- `disconnect`

Please note that since Socket.IO v3, the Socket instance does not emit any event related to the reconnection logic anymore. You can listen to the events on the Manager instance directly:

```
socket.io.on("reconnect_attempt", () => {
  // ...
});

socket.io.on("reconnect", () => {
  // ...
});
```

More information can be found in the migration guide.

`connect`

This event is fired by the Socket instance upon connection **and** reconnection.

```
socket.on("connect", () => {
  // ...
});
```

Please note that you shouldn't register event handlers in the `connect` handler itself, as a new handler will be registered every time the Socket reconnects:

```
// BAD
socket.on("connect", () => {
  socket.on("data", () => { /* ... */ });
});

// GOOD
socket.on("connect", () => {
  // ...
});

socket.on("data", () => { /* ... */ });
```

`connect_error`

This event is fired when:

- the low-level connection cannot be established
- the connection is denied by the server in a middleware function

In the first case, the Socket will automatically try to reconnect, after a given delay.

In the latter case, you need to manually reconnect. You might need to update the credentials:

```
// either by directly modifying the `auth` attribute
socket.on("connect_error", () => {
  socket.auth.token = "abcd";
  socket.connect();
});

// or if the `auth` attribute is a function
const socket = io({
  auth: (cb) => {
    cb(localStorage.getItem("token"));
  }
});

socket.on("connect_error", () => {
  setTimeout(() => {
    socket.connect();
  }, 1000);
});
```

`disconnect`

This event is fired upon disconnection.

```
socket.on("disconnect", (reason) => {
  // ...
});
```

Here is the list of possible reasons:

Reason	Description
`io server disconnect`	The server has forcefully disconnected the socket with socket.disconnect()
`io client disconnect`	The socket was manually disconnected using socket.disconnect()

Reason	Description
`ping timeout`	The server did not send a PING within the `pingInterval + pingTimeout` range
`transport close`	The connection was closed (example: the user has lost connection, or the network was changed from WiFi to 4G)
`transport error`	The connection has encountered an error (example: the server was killed during a HTTP long-polling cycle)

In the first two cases (explicit disconnection), the client will not try to reconnect and you need to manually call `socket.connect()`.

In all other cases, the client will wait for a small random delay and then try to reconnect:

```
socket.on("disconnect", (reason) => {
  if (reason === "io server disconnect") {
    // the disconnection was initiated by the server, you need to reconnect
manually
    socket.connect();
  }
  // else the socket will automatically try to reconnect
});
```

Note: those events, along with `disconnecting`, `newListener` and `removeListener`, are special events that shouldn't be used in your application:

```
// BAD, will throw an error
socket.emit("disconnect");
```

Complete API

The complete API exposed by the Socket instance can be found here.

Edit this page

*Last updated on **12/19/2022***