

Version: 4.x[On this page](#)

Listening to events

There are several ways to handle events that are transmitted between the server and the client.

EventEmitter methods

On the server-side, the `Socket` instance extends the Node.js [EventEmitter](#) class.

On the client-side, the `Socket` instance uses the event emitter provided by the [component-emitter](#) library, which exposes a subset of the `EventEmitter` methods.

`socket.on(eventName, listener)`

Adds the *listener* function to the end of the listeners array for the event named *eventName*.

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

`socket.once(eventName, listener)`

Adds a **one-time** *listener* function for the event named *eventName*

```
socket.once("details", (...args) => {  
  // ...  
});
```

`socket.off(eventName, listener)`

Removes the specified *listener* from the listener array for the event named *eventName*.

```
const listener = (...args) => {
  console.log(args);
}

socket.on("details", listener);

// and then later...
socket.off("details", listener);
```

socket.removeAllListeners([eventName])

Removes all listeners, or those of the specified *eventName*.

```
// for a specific event
socket.removeAllListeners("details");
// for all events
socket.removeAllListeners();
```

Catch-all listeners

Since Socket.IO v3, a new API inspired from the [EventEmitter2](#) library allows to declare catch-all listeners.

This feature is available on both the client and the server.

socket.onAny(listener)

Adds a listener that will be fired when any event is emitted.

```
socket.onAny((eventName, ...args) => {
  // ...
});
```

socket.prependAny(listener)

Adds a listener that will be fired when any event is emitted. The listener is added to the beginning of the listeners array.

```
socket.prependAny((eventName, ...args) => {
  // ...
});
```

socket.offAny([listener])

Removes all catch-all listeners, or the given listener.

```
const listener = (eventName, ...args) => {
  console.log(eventName, args);
}

socket.onAny(listener);

// and then later...
socket.offAny(listener);

// or all listeners
socket.offAny();
```

Validation

The validation of the event arguments is out of the scope of the Socket.IO library.

There are many packages in the JS ecosystem which cover this use case, among them:

- [joi](#)
- [ajv](#)
- [validatorjs](#)

Example with [joi](#) and [acknowledgements](#):

```
const Joi = require("joi");

const userSchema = Joi.object({
  username: Joi.string().max(30).required(),
  email: Joi.string().email().required()
});

io.on("connection", (socket) => {
```

```

socket.on("create user", (payload, callback) => {
  if (typeof callback !== "function") {
    // not an acknowledgement
    return socket.disconnect();
  }
  const { error, value } = userSchema.validate(payload);
  if (error) {
    return callback({
      status: "KO",
      error
    });
  }
  // do something with the value, and then
  callback({
    status: "OK"
  });
});

```

});

Error handling

There is currently no built-in error handling in the Socket.IO library, which means you must catch any error that could be thrown in a listener.

```

io.on("connection", (socket) => {
  socket.on("list items", async (callback) => {
    try {
      const items = await findItems();
      callback({
        status: "OK",
        items
      });
    } catch (e) {
      callback({
        status: "NOK"
      });
    }
  });
});

```

This can be refactored into:

```
const errorHandler = (handler) => {
  const handleError = (err) => {
    console.error("please handle me", err);
  };

  return (...args) => {
    try {
      const ret = handler.apply(this, args);
      if (ret && typeof ret.catch === "function") {
        // async handler
        ret.catch(handleError);
      }
    } catch (e) {
      // sync handler
      handleError(e);
    }
  };
};

// server or client side
socket.on("hello", errorHandler(() => {
  throw new Error("let's panic");
}));
```

 [Edit this page](#)

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