

License

[MIT](#)

ws: a Node.js WebSocket library

[Coverage Status](#)

ws is a simple to use, blazing fast, and thoroughly tested WebSocket client and server implementation.

Passes the quite extensive Autobahn test suite: [server](#), [client](#).

Note: This module does not work in the browser. The client in the docs is a reference to a backend with the role of a client in the WebSocket communication. Browser clients must use the native [WebSocket](#) object. To make the same code work seamlessly on Node.js and the browser, you can use one of the many wrappers available on npm, like [isomorphic-ws](#).

Table of Contents

[Protocol support](#)

[Installing](#)

[Opt-in for performance](#)

[Legacy opt-in for performance](#)

[API docs](#)

[WebSocket compression](#)

[Usage examples](#)

[Sending and receiving text data](#)

```

        // instead of `Websocket#close()`,  

        // which waits for the close timer.  

        // Delay should be equal to the  

        // interval at which your server  

        // sends out pings plus a conservative  

        // assumption of the latency.  

        Client authentication  

        Multiple servers sharing a single HTTP/S server  

        Extreme HTTP/S server  

        Simple server  

        Sending binary data  

        Client authentication  

        Round-trip time  

        Server broadcast  

        Use the Node.js streams API  

        Other examples  

        FAQ  

        How to get the IP address of the client?  

        How to detect and close broken connections?  

        How to connect via a proxy?  

        ChangeLog  

        License
    
```

const client = new Websocket('ws://websocket-echo.com/');

{
 this.pingTimeout = setTimeout(() => {
 this.terminate();
 }, 30000 + 1000);

 client.onerror = error => {
 client.onerror = clearPingTimeout;
 console.error(error);
 };
 client.onopen = open => {
 client.onpong = ping => {
 heartbeat();
 };
 };
 client.onclose = close => {
 clearPingTimeout();
 if (close.code === 1000) {
 heartbeat();
 }
 };
}

- [How to connect via a proxy?](#)
 - [FAQ](#)
 - [How to detect and close broken connections?](#)
 - [How to connect via a proxy?](#)
 - [ChangeLog](#)
 - [License](#)
- (8) [HYBi drafts 07-12](#) (Use the option protocolVersion:
protocolVersion: 13)
- [HYBi drafts 13-17](#) (Current default, alternatively option
agent or socks-proxy-agent.
- Use a custom http. Agent implementation like https-proxy-

ChangeLog

```

});;

const interval = setInterval(function
  ping() {
    wss.clients.forEach(function each(ws)
    {
      if (ws.isAlive === false) return
      ws.terminate();

      ws.isAlive = false;
      ws.ping();
    });
}, 30000);

wss.on('close', function close() {
  clearInterval(interval);
});

```

Pong messages are automatically sent in response to ping messages as required by the spec.

Just like the server example above, your clients might as well lose connection without knowing it. You might want to add a ping listener on your clients to prevent that. A simple implementation would be:

```

import WebSocket from 'ws';

function heartbeat() {
  clearTimeout(this.pingTimeout);

  // Use `WebSocket#terminate()`, which
    // immediately destroys the
      // connection,

```

Installing

`npm install ws`

Opt-in for performance

[bufferutil](#) is an optional module that can be installed alongside the ws module:

`npm install --save-optional bufferutil`

This is a binary addon that improves the performance of certain operations such as masking and unmasking the data payload of the WebSocket frames. Prebuilt binaries are available for the most popular platforms, so you don't necessarily need to have a C++ compiler installed on your machine.

To force ws to not use bufferutil, use the [WS_NO_BUFFER_UTIL](#) environment variable. This can be useful to enhance security in systems where a user can put a package in the package search path of an application of another user, due to how the Node.js resolver algorithm works.

Legacy opt-in for performance

If you are running on an old version of Node.js (prior to v18.14.0), ws also supports the [utf-8-validate](#) module:

`npm install --save-optional utf-8-validate`

```

ws.on('pong', heartbeat);
ws.on('error', console.error);

ws.isAlive = true;
connection.ws = {
  connection,
  function ws() {
    const ws = new WebSocketServer({ port: 8080 });
    ws.on('connection', function connection(ws) {
      ws.on('pong', heartbeat);
      ws.on('error', console.error);
      ws.isAlive = true;
      function heartbeat() {
        ws.send('ping');
      }
      ws.on('error', function error(error) {
        if (error.message === 'Connection closed by peer') {
          ws.close();
        }
      });
    });
  }
}

```

In these cases, ping messages can be used as a means to verify that the remote endpoint is still responsive.

Sometimes, the link between the server and the client can be interrupted in a way that keeps both the server and the client unaware of the broken state of the connection (e.g. when pulling the cord).

How to detect and close broken connections?

```

ws.on('error', console.error);

const ip = req.headers['x-forwarded-for'].split(',')[0].trim();
for (let i = 0; i < ip.length; i++) {
  ws.on(`connection${i}`, function connection(ws) {
    ws.on('error', console.error);
  });
}

```

See [/doc/ws.md](#) for Node.js-like documentation of ws classes and utility functions.

API docs

To force ws not to use utf-8-validate, use the [WS_NO_UTF_8_VALIDATE](#) environment variable.

This contains a binary polyfill for [buffer.isUtf8\(\)](#).

WebSocket compression

ws supports the [permessage-deflate extension](#) which enables the client and server to negotiate a compression algorithm and its parameters, and then selectively apply it to the data payloads of each WebSocket message.

The extension is disabled by default on the server and enabled by default on the client. It adds a significant overhead in terms of performance and memory consumption so we suggest to enable it only if it is really needed.

Note that Node.js has a variety of issues with high-performance compression, where increased concurrency, especially on Linux, can lead to [catastrophic memory fragmentation](#) and slow performance. If you intend to use [permessage-deflate](#) in production, it is worthwhile to set up a test environment and see how it performs.

Other examples

For a full example with a browser client communicating with a ws server, see the examples folder.

Otherwise, see the test cases.

FAQ

How to get the IP address of the client?

The remote IP address can be obtained from the raw socket.

```
import { WebSocketServer } from 'ws';

const wss = new WebSocketServer({ port:
  8080 });

wss.on('connection', function
  connection(ws, req) {
  const ip = req.socket.remoteAddress;

  ws.on('error', console.error);
});
```

When the server runs behind a proxy like NGINX, the de-facto standard is to use the X-Forwarded-For header.

representative of your workload and ensure Node.js/zlib will handle it with acceptable performance and memory usage.

Tuning of permessage-deflate can be done via the options defined below. You can also use `zlibDeflateOptions` and `zlibInflateOptions`, which is passed directly into the creation of [raw deflate/inflate streams](#).

See [the docs](#) for more options.

```
import WebSocket, { WebSocketServer }
  from 'ws';

const wss = new WebSocketServer({
  port: 8080,
  perMessageDeflate: {
    zlibDeflateOptions: {
      // See zlib defaults.
      chunkSize: 1024,
      memLevel: 7,
      level: 3
    },
    zlibInflateOptions: {
      chunkSize: 10 * 1024
    },
    // Other options settable:
    clientNoContextTakeover: true, // Defaults to negotiated value.
    serverNoContextTakeover: true, // Defaults to negotiated value.
    serverMaxWindowBits: 10, // Defaults to negotiated value.
    // Below options specified as default values.
```

```
const ws = new WebSocket('ws://'
import WebSocket from 'ws';

www.host.com/path');

Send and receiving text data
```

Usage examples

```
process.stdin.pipe(duplex);
duplex.pipe(process.stdout);

duplex.on('error', console.error);

encoding: 'utf8', {}));
createWebSocketStream(ws, {
const duplex =
    www.host.echo.com/';

const ws = new WebSocket('ws://'
    'ws';

} createWebSocketStream { from
import WebSocket,
```

Use the Node.js streams API

```
)());
{}),
{, 500);
ws.send(Date.now());
setTimout(function timeout() {
    ws.on('message', function message(data) {
        console.log(`Round-trip time: ${Date.now() - data} ms`);
    });
}, // Limits zlib concurrency
concurrencyLimit: 10, // Limit to 10 messages
threshold: 1024 // Size (in bytes)
for perf.
below which messages
should not be compressed if
context takeover is disabled.
The client will only use the extension if it is supported and
enabled on the server. To always disable the extension on the
client, set the perMessageDeflate option to false.
```

```
perMessageDeflate: false
www.host.com/path',
const ws = new WebSocket('ws://
import WebSocket from 'ws';

());
}

// Should not be compressed if
// threshold: 1024 // Size (in bytes)
// for perf.
// Below which messages
// should not be compressed if
// context takeover is disabled.
The client will only use the extension if it is supported and
enabled on the server. To always disable the extension on the
client, set the perMessageDeflate option to false.
```

```

ws.on('message', function
  message(data, isBinary) {
    wss.clients.forEach(function
      each(client) {
        if (client !== ws &&
          client.readyState ===
          WebSocket.OPEN) {
          client.send(data, { binary:
            isBinary });
        }
      });
    });
  });
}
);

```

Round-trip time

```

import WebSocket from 'ws';

const ws = new WebSocket('wss://
  websocket-echo.com/');

ws.on('error', console.error);

ws.on('open', function open() {
  console.log('connected');
  ws.send(Date.now());
});

ws.on('close', function close() {
  console.log('disconnected');
});

```

```

ws.on('error', console.error);

ws.on('open', function open() {
  ws.send('something');
});

ws.on('message', function message(data)
  {
    console.log('received: %s', data);
  });

```

Sending binary data

```

import WebSocket from 'ws';

const ws = new WebSocket('ws://
  www.host.com/path');

ws.on('error', console.error);

ws.on('open', function open() {
  const array = new Float32Array(5);

  for (var i = 0; i < array.length; +
    +i) {
    array[i] = i / 2;
  }

  ws.send(array);
});

```

```

const ws = new WebSocketServer({ port: 8080 });

ws.on('connection', function(ws) {
  ws.on('message', function(data) {
    ws.clients.forEach(function(client) {
      if (client.readyState === WebSocket.OPEN) {
        client.send(data);
      }
    });
  });
});

ws.on('error', console.error);

```

A client WebSocket broadcasting to every other connected WebSocket clients, excluding itself.

Websocket A client WebSocket broadcasting to every other connected

```

import { WebSocket } from 'ws';

const ws = new WebSocketServer({ port: 8080 });

ws.on('connection', function(ws) {
  ws.on('message', function(data) {
    ws.clients.forEach(function(client) {
      if (client.readyState === WebSocket.OPEN) {
        client.send(data);
      }
    });
  });
});

ws.on('error', console.error);

```

Connection Websocket connection function

WSA WS on('error', console.error);

```

const server = createServer();
server.listen(8080, () => {
  console.log(`HTTP server listening on port 8080`);
});

```

```

import { createServer } from 'https';
import { readFileSync } from 'fs';
import { readFileSync } from 'path/to/key.pem';

```

External HTTP/S server

```

ws.send('something');
}

```

```

ws.on('message', function(data) {
  console.log(`received: ${data}`);
});

```

```

ws.on('error', console.error);
connection(ws) {
  ws.on('message', function(data) {
    message(data);
  });
}

```

```

const ws = new WebSocketServer({ port: 8080 });

ws.on('connection', function(ws) {
  ws.on('message', function(data) {
    ws.clients.forEach(function(client) {
      if (client.readyState === WebSocket.OPEN) {
        client.send(data);
      }
    });
  });
});

ws.on('error', console.error);

```

Simple server

```

authenticate(request, function
  next(err, client) {
  if (err || !client) {
    socket.write('HTTP/1.1 401
      Unauthorized\r\n\r\n');
    socket.destroy();
    return;
  }
  socket.removeListener('error',
    onSocketError);

  wss.handleUpgrade(request, socket,
    head, function done(ws) {
    wss.emit('connection', ws,
      request, client);
  });
});
});

server.listen(8080);

```

Also see the provided [example](#) using express-session.

Server broadcast

A client WebSocket broadcasting to all connected WebSocket clients, including itself.

```

import WebSocket, { WebSocketServer }
  from 'ws';

```

```

const wss = new
  WebSocketServer({ server });

wss.on('connection', function
  connection(ws) {
  ws.on('error', console.error);

  ws.on('message', function
    message(data) {
    console.log('received: %s', data);
  });

  ws.send('something');
});

server.listen(8080);

```

Multiple servers sharing a single HTTP/S server

```

import { createServer } from 'http';
import { WebSocketServer } from 'ws';

const server = createServer();
const wss1 = new WebSocketServer({
  noServer: true });
const wss2 = new WebSocketServer({
  noServer: true });

wss1.on('connection', function
  connection(ws) {
  ws.on('error', console.error);

```

Client authentication

```
import { createServer } from 'http';
import { WebSocketServer } from 'ws';

const server = createServer();
server.on('upgrade', function(ws, request) {
    ws.on('connection', function(ws) {
        ws.on('error', console.error);
        ws.on('message', function(message) {
            console.log(`Received message ${message}`);
        });
    });
});

const ws = new WebSocketServer({ noServer: true });
ws.on('connection', function(ws) {
    ws.on('error', console.error);
    ws.on('message', function(message) {
        console.log(`Received message ${message}`);
    });
});
```

11

```
// ...
ws2.on('connection', function(ws) {
    connect(ws);
    ws.on('error', console.error);
    ws.on('message', function(message) {
        console.log(`Received message ${message}`);
    });
});

const connect = (ws) => {
    ws.on('error', console.error);
    ws.on('upgrade', function(request, socket, head) {
        if (request.url === '/foo') {
            ws1.handleUpgrade(request, socket, head, function done(ws) {
                ws1.emit('connection', ws);
            });
        } else if (request.url === '/bar') {
            ws2.handleUpgrade(request, socket, head, function done(ws) {
                ws2.emit('connection', ws);
            });
        }
    });
};

function handleUpgrade(ws, socket, head, fn) {
    const pathname = new URL(ws.url).pathname;
    if (pathname === '/foo') {
        ws1.upgrade(ws, socket, head);
    } else if (pathname === '/bar') {
        ws2.upgrade(ws, socket, head);
    }
    fn();
}

function upgrade(ws, socket, head) {
    const pathname = new URL(ws.url).pathname;
    if (pathname === '/foo') {
        ws1.emit('connection', ws);
    } else if (pathname === '/bar') {
        ws2.emit('connection', ws);
    }
}
```

10