# WebTorrent



# WebTorrent Documentation

WebTorrent is a streaming torrent client for **Node.js** and the **web**. WebTorrent provides the same API in both environments.

To use WebTorrent in the browser, WebRTC support is required (Chrome, Firefox, Opera).

## Install

```
npm install webtorrent
```

## Quick Example

```
var client = new WebTorrent()

var torrentId = 'magnet:?
xt=urn:btih:6a9759bffd5c0af65319979fb7832189f4f3c35d&dn=sintel.mp4&tr=wss%3A%2F%2Ftr
acker.btorrent.xyz&tr=wss%3A%2F%2Ftracker.fastcast.nz&tr=wss%3A%2F%2Ftracker.openweb
torrent.com&ws=https%3A%2F%2Fwebtorrent.io%2Ftorrents%2Fsintel-1024-surround.mp4'

client.add(torrentId, function (torrent) {
  // Torrents can contain many files. Let's use the first.
  var file = torrent.files[0]

  // Display the file by adding it to the DOM. Supports video, audio, image, etc.
files
  file.appendTo('body')
})
```

# WebTorrent API

## WebTorrent.WEBRTC_SUPPORT

Is WebRTC natively supported in the environment?

```
if (WebTorrent.WEBRTC_SUPPORT) {
  // WebRTC is supported
} else {
```

```
  // Use a fallback
}
```

# client = new WebTorrent([opts])

Create a new `WebTorrent` instance.

If `opts` is specified, then the default options (shown below) will be overridden.

```
{
  maxConns: Number,        // Max number of connections per torrent (default=55)
  nodeId: String|Buffer,   // DHT protocol node ID (default=randomly generated)
  peerId: String|Buffer,   // Wire protocol peer ID (default=randomly generated)
  tracker: Boolean|Object, // Enable trackers (default=true), or options object for
Tracker
  dht: Boolean|Object,     // Enable DHT (default=true), or options object for DHT
  webSeeds: Boolean        // Enable BEP19 web seeds (default=true)
}
```

For possible values of `opts.dht` see the `bittorrent-dht` documentation.

For possible values of `opts.tracker` see the `bittorrent-tracker` documentation.

# client.add(torrentId, [opts], [function ontorrent (torrent) {}])

Start downloading a new torrent.

`torrentId` can be one of:

- magnet uri (string)
- torrent file (buffer)
- info hash (hex string or buffer)
- parsed torrent (from parse-torrent)
- http/https url to a torrent file (string)
- filesystem path to a torrent file (string) *(Node.js only)*

If `opts` is specified, then the default options (shown below) will be overridden.

```
{
  announce: [],             // Torrent trackers to use (added to list in .torrent
or magnet uri)
  getAnnounceOpts: Function, // Custom callback to allow sending extra parameters to
the tracker
  maxWebConns: Number,      // Max number of simultaneous connections per web seed
[default=4]
  path: String,             // Folder to download files to
(default=`/tmp/webtorrent/`)
  store: Function           // Custom chunk store (must follow [abstract-chunk-
```

```
store](https://www.npmjs.com/package/abstract-chunk-store) API)
}
```

If `ontorrent` is specified, then it will be called when **this** torrent is ready to be used (i.e. metadata is available). Note: this is distinct from the 'torrent' event which will fire for **all** torrents.

If you want access to the torrent object immediately in order to listen to events as the metadata is fetched from the network, then use the return value of `client.add`. If you just want the file data, then use `ontorrent` or the 'torrent' event.

If you provide `opts.store`, it will be called as `opts.store(chunkLength, storeOpts)` with:

- `storeOpts.length` - size of all the files in the torrent
- `storeOpts.files` - an array of torrent file objects
- `storeOpts.torrent` - the torrent instance being stored

# `client.seed(input, [opts], [function onseed (torrent) {}])`

Start seeding a new torrent.

`input` can be any of the following:

- filesystem path to file or folder (string) *(Node.js only)*
- W3C File object (from an `<input>` or drag and drop) *(browser only)*
- W3C FileList object (basically an array of `File` objects) *(browser only)*
- Node Buffer object
- Node Readable stream object

Or, an **array of `string`, `File`, `Buffer`, or `stream.Readable` objects**.

If `opts` is specified, it should contain the following types of options:

- options for create-torrent (to allow configuration of the .torrent file that is created)
- options for `client.add` (see above)

If `onseed` is specified, it will be called when the client has begun seeding the file.

**Note:** Every torrent is required to have a name. If one is not explicitly provided through `opts.name`, one will be determined automatically using the following logic:

- If all files share a common path prefix, that will be used. For example, if all file paths start with `/imgs/` the torrent name will be `imgs`.
- Otherwise, the first file that has a name will determine the torrent name. For example, if the first file is `/foo/bar/baz.txt`, the torrent name will be `baz.txt`.
- If no files have names (say that all files are Buffer or Stream objects), then a name like "Unnamed Torrent " will be generated.

**Note:** Every file is required to have a name. For filesystem paths or W3C File objects, the name is included in the object. For Buffer or Readable stream types, a `name` property can be set on the object, like this:

```
var buf = new Buffer('Some file content')
buf.name = 'Some file name'
client.seed(buf, cb)
```

## client.on('torrent', function (torrent) {})

Emitted when a torrent is ready to be used (i.e. metadata is available and store is ready). See the torrent section for more info on what methods a `torrent` has.

## client.on('error', function (err) {})

Emitted when the client encounters a fatal error. The client is automatically destroyed and all torrents are removed and cleaned up when this occurs.

Always listen for the 'error' event.

## client.remove(torrentId, [function callback (err) {}])

Remove a torrent from the client. Destroy all connections to peers and delete all saved file data. If `callback` is specified, it will be called when file data is removed.

## client.destroy([function callback (err) {}])

Destroy the client, including all torrents and connections to peers. If `callback` is specified, it will be called when the client has gracefully closed.

## client.torrents[...]

An array of all torrents in the client.

## client.get(torrentId)

Returns the torrent with the given `torrentId`. Convenience method. Easier than searching through the `client.torrents` array. Returns `null` if no matching torrent found.

## client.downloadSpeed

Total download speed for all torrents, in bytes/sec.

### `client.uploadSpeed`

Total upload speed for all torrents, in bytes/sec.

### `client.progress`

Total download progress for all **active** torrents, from 0 to 1.

### `client.ratio`

Aggregate "seed ratio" for all torrents (uploaded / downloaded), from 0 to 1.

## Torrent API

### `torrent.infoHash`

Info hash of the torrent (string).

### `torrent.magnetURI`

Magnet URI of the torrent (string).

### `torrent.torrentFile`

`.torrent` file of the torrent (Buffer).

### `torrent.torrentFileBlobURL` *(browser only)*

`.torrent` file of the torrent (Blob URL).

### `torrent.files[...]`

Array of all files in the torrent. See documentation for `File` below to learn what methods/properties files have.

### `torrent.timeRemaining`

Time remaining for download to complete (in milliseconds).

## torrent.received

Total bytes received from peers (*including* invalid data).

## torrent.downloaded

Total *verified* bytes received from peers.

## torrent.uploaded

Total bytes uploaded to peers.

## torrent.downloadSpeed

Torrent download speed, in bytes/sec.

## torrent.uploadSpeed

Torrent upload speed, in bytes/sec.

## torrent.progress

Torrent download progress, from 0 to 1.

## torrent.ratio

Torrent "seed ratio" (uploaded / downloaded), from 0 to 1.

## torrent.numPeers

Number of peers in the torrent swarm.

## torrent.path

Torrent download location.

## torrent.destroy([callback])

Alias for `client.remove(torrent)`. If `callback` is provided, it will be called when the torrent is fully destroyed, i.e. all open sockets are closed, and the storage is closed.

## torrent.addPeer(peer)

Add a peer to the torrent swarm. This is advanced functionality. Normally, you should not need to call `torrent.addPeer()` manually. WebTorrent will automatically find peers using the tracker servers or DHT. This is just for manually adding a peer to the client.

This method should not be called until the `infoHash` event has been emitted.

Returns `true` if peer was added, `false` if peer was blocked by the loaded blocklist.

The `peer` argument must be an address string in the format `12.34.56.78:4444` (for normal TCP/uTP peers), or a `simple-peer` instance (for WebRTC peers).

## torrent.addWebSeed(url)

Add a web seed to the torrent swarm. For more information on BitTorrent web seeds, see BEP19.

In the browser, web seed servers must have proper CORS (Cross-origin resource sharing) headers so that data can be fetched across domain.

The `url` argument is the web seed URL.

## torrent.removePeer(peer)

Remove a peer from the torrent swarm. This is advanced functionality. Normally, you should not need to call `torrent.removePeer()` manually. WebTorrent will automatically remove peers from the torrent swarm when they're slow or don't have pieces that are needed.

The `peer` argument should be an address (i.e. "ip:port" string), a peer id (hex string), or `simple-peer` instance.

## torrent.select(start, end, [priority], [notify])

Selects a range of pieces to prioritize starting with `start` and ending with `end` (both inclusive) at the given `priority`. `notify` is an optional callback to be called when the selection is updated with new data.

## torrent.deselect(start, end, priority)

Deprioritizes a range of previously selected pieces.

# torrent.critical(start, end)

Marks a range of pieces as critical priority to be downloaded ASAP. From `start` to `end` (both inclusive).

# torrent.createServer([opts])

Create an http server to serve the contents of this torrent, dynamically fetching the needed torrent pieces to satisfy http requests. Range requests are supported.

Returns an `http.Server` instance (got from calling `http.createServer`). If `opts` is specified, it is passed to the `http.createServer` function.

Visiting the root of the server `/` will show a list of links to individual files. Access individual files at `/<index>` where `<index>` is the index in the `torrent.files` array (e.g. `/0`, `/1`, etc.)

Here is a usage example:

```
var client = new WebTorrent()
var magnetURI = 'magnet: ...'

client.add(magnetURI, function (torrent) {
  // create HTTP server for this torrent
  var server = torrent.createServer()
  server.listen(port) // start the server listening to a port

  // visit http://localhost:<port>/ to see a list of files

  // access individual files at http://localhost:<port>/<index> where index is the
index
  // in the torrent.files array

  // later, cleanup...
  server.close()
  client.destroy()
})
```

# torrent.pause()

Temporarily stop connecting to new peers. Note that this does not pause new incoming connections, nor does it pause the streams of existing connections or their wires.

# torrent.resume()

Resume connecting to new peers.

# torrent.on('infoHash', function () {})

Emitted when the info hash of the torrent has been determined.

## torrent.on('metadata', function () {})

Emitted when the metadata of the torrent has been determined. This includes the full contents of the .torrent file, including list of files, torrent length, piece hashes, piece length, etc.

## torrent.on('ready', function () {})

Emitted when the torrent is ready to be used (i.e. metadata is available and store is ready).

## torrent.on('warning', function (err) {})

Emitted when there is a warning. This is purely informational and it is not necessary to listen to this event, but it may aid in debugging.

## torrent.on('error', function (err) {})

Emitted when the torrent encounters a fatal error. The torrent is automatically destroyed and removed from the client when this occurs.

**Note:** Torrent errors are emitted at `torrent.on('error')`. If there are no 'error' event handlers on the torrent instance, then the error will be emitted at `client.on('error')`. This prevents throwing an uncaught exception (unhandled 'error' event), but it makes it impossible to distinguish client errors versus torrent errors. Torrent errors are not fatal, and the client is still usable afterwards. Therefore, always listen for errors in both places ( `client.on('error')` and `torrent.on('error')` ).

## torrent.on('done', function () {})

Emitted when all the torrent files have been downloaded.

Here is a usage example:

```
torrent.on('done', function(){
  console.log('torrent finished downloading');
  torrent.files.forEach(function(file){
     // do something with file
  })
})
```

## torrent.on('download', function (bytes) {})

Emitted whenever data is downloaded. Useful for reporting the current torrent status, for instance:

```
torrent.on('download', function (bytes) {
  console.log('just downloaded: ' + bytes)
  console.log('total downloaded: ' + torrent.downloaded);
  console.log('download speed: ' + torrent.downloadSpeed)
  console.log('progress: ' + torrent.progress)
})
```

## torrent.on('upload', function (bytes) {})

Emitted whenever data is uploaded. Useful for reporting the current torrent status.

## torrent.on('wire', function (wire) {})

Emitted whenever a new peer is connected for this torrent. `wire` is an instance of `bittorrent-protocol`, which is a node.js-style duplex stream to the remote peer. This event can be used to specify custom BitTorrent protocol extensions.

Here is a usage example:

```
var MyExtension = require('./my-extension')

torrent1.on('wire', function (wire, addr) {
  console.log('connected to peer with address ' + addr)
  wire.use(MyExtension)
})
```

See the `bittorrent-protocol` extension api docs for more information on how to define a protocol extension.

## torrent.on('noPeers', function (announceType) {})

Emitted whenever a DHT or tracker announce occurs, but no peers have been found. `announceType` is either `'tracker'` or `'dht'` depending on which announce occurred to trigger this event. Note that if you're attempting to discover peers from both a tracker and a DHT, you'll see this event separately for each.

# File API

## file.name

File name, as specified by the torrent. *Example: 'some-filename.txt'*

## file.path

File path, as specified by the torrent. *Example: 'some-folder/some-filename.txt'*

## file.length

File length (in bytes), as specified by the torrent. *Example: 12345*

## file.downloaded

Total *verified* bytes received from peers, for this file.

## file.select()

Selects the file to be downloaded, but at a lower priority than files with streams. Useful if you know you need the file at a later stage.

## file.deselect()

Deselects the file, which means it won't be downloaded unless someone creates a stream for it.

## stream = file.createReadStream([opts])

Create a readable stream to the file. Pieces needed by the stream will be prioritized highly and fetched from the swarm first.

You can pass `opts` to stream only a slice of a file.

```
{
  start: startByte,
  end: endByte
}
```

Both `start` and `end` are inclusive.

## file.getBuffer(function callback (err, buffer) {})

Get the file contents as a `Buffer`.

The file will be fetched from the network with highest priority, and `callback` will be called once the file is ready. `callback` must be specified, and will be called with a an `Error` (or `null`) and the file contents as a

`Buffer`.

```
file.getBuffer(function (err, buffer) {
  if (err) throw err
  console.log(buffer) // <Buffer 00 98 00 01 ...>
})
```

# `file.appendTo(rootElem, [opts], [function callback (err, elem) {}])` *(browser only)*

Show the file in a the browser by appending it to the DOM. This is a powerful function that handles many file types like video (.mp4, .webm, .m4v, etc.), audio (.m4a, .mp3, .wav, etc.), images (.jpg, .gif, .png, etc.), and other file formats (.pdf, .md, .txt, etc.).

The file will be fetched from the network with highest priority and streamed into the page (if it's video or audio). In some cases, video or audio files will not be streamable because they're not in a format that the browser can stream so the file will be fully downloaded before being played. For other non-streamable file types like images and PDFs, the file will be downloaded then displayed.

`rootElem` is a container element (CSS selector or reference to DOM node) that the content will be shown in. A new DOM node will be created for the content and appended to `rootElem`.

If provided, `opts` can contain the following options:

- `autoplay`: Autoplay video/audio files (default: `true`)
- `controls`: Show video/audio player controls (default: `true`)
- `maxBlobLength`: Files above this size will skip the "blob" strategy and fail (default: `200 * 1000 * 1000` bytes)

If provided, `callback` will be called once the file is visible to the user. `callback` is called with an `Error` (or `null`) and the new DOM node that is displaying the content.

```
file.appendTo('#containerElement', function (err, elem) {
  if (err) throw err // file failed to download or display in the DOM
  console.log('New DOM node with the content', elem)
})
```

Streaming support depends on support for `MediaSource` API in the browser. All modern browsers have `MediaSource` support.

For video and audio, webtorrent tries multiple methods of playing the file:

- `videostream` -- best option, supports streaming **with seeking**, but only works with MP4-based files for now (uses `MediaSource` API)
- `mediasource` -- supports more formats, supports streaming **without seeking** (uses `MediaSource` API)
- Blob URL -- supports the most formats of all (anything the `<video>` tag supports from an http url), **with seeking**, but **does not support streaming** (entire file must be downloaded first)

The Blob URL strategy will not be attempted if the file is over `opts.maxBlobLength` (200 MB by default) since it requires the entire file to be downloaded before playback can start which gives the appearance of the `<video>` tag being stalled. If you increase the size, be sure to indicate loading progress to the user in the UI somehow.

For other media formats, like images, the file is just added to the DOM.

For text-based formats, like html files, pdfs, etc., the file is added to the DOM via a sandboxed `<iframe>` tag.

## `file.renderTo(elem, [opts], [function callback (err, elem) {}])` *(browser only)*

Like `file.appendTo` but renders directly into given element (or CSS selector).

## `file.getBlob(function callback (err, blob) {})` *(browser only)*

Get a W3C `Blob` object which contains the file data.

The file will be fetched from the network with highest priority, and `callback` will be called once the file is ready. `callback` must be specified, and will be called with a an `Error` (or `null`) and the `Blob` object.

## `file.getBlobURL(function callback (err, url) {})` *(browser only)*

Get a url which can be used in the browser to refer to the file.

The file will be fetched from the network with highest priority, and `callback` will be called once the file is ready. `callback` must be specified, and will be called with a an `Error` (or `null`) and the Blob URL (`String`).

This method is useful for creating a file download link, like this:

```
file.getBlobURL(function (err, url) {
  if (err) throw err
  var a = document.createElement('a')
  a.download = file.name
  a.href = url
  a.textContent = 'Download ' + file.name
  document.body.appendChild(a)
})
```

## Learn

Get Started
Docs
FAQ

## Connect

GitHub
Twitter

## Support

Open an issue
Chat in Gitter
**#webtorrent** on freenode (IRC)

## Star on GitHub