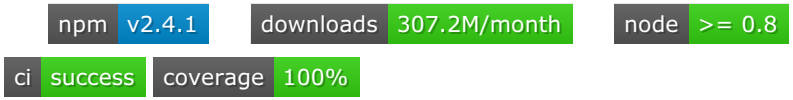
on-finished



Execute a callback when a HTTP request closes, finishes, or errors.

Install

This is a Node.js module available through the npm registry. Installation is done using the npm install command:

```
$ npm install on-finished
```

API

```
var onFinished = require('on-finished')
```

onFinished(res, listener)

Attach a listener to listen for the response to finish. The listener will be invoked only once when the response finished. If

the response finished to an error, the first argument will contain the error. If the response has already finished, the listener will be invoked.

Listening to the end of a response would be used to close things associated with the response, like open files.

Listener is invoked as `listener(err, res)`.

```
onFinished(res, function (err, res) {
  // clean up open fds, etc.
  // err contains the error if request
       error'd
})
```

onFinished(req, listener)

Attach a listener to listen for the request to finish. The listener will be invoked only once when the request finished. If the request finished to an error, the first argument will contain the error. If the request has already finished, the listener will be invoked.

Listening to the end of a request would be used to know when to continue after reading the data.

Listener is invoked as `listener(err, req)`.

```
var data = ''

req.setEncoding('utf8')
req.on('data', function (str) {
  data += str
})
```

License

MIT


```
onFinished(req, function (err, req) {
  // data is read unless there is err
})
```

onFinished.isFinished(res)

Determine if `res` is already finished. This would be useful to check and not even start certain operations if the response has already finished.

onFinished.isFinished(req)

Determine if `req` is already finished. This would be useful to check and not even start certain operations if the request has already finished.

Special Node.js requests

HTTP CONNECT method

The meaning of the `CONNECT` method from RFC 7231, section 4.3.6:

> *The CONNECT method requests that the recipient establish a tunnel to the destination origin server identified by the request-target and, if successful, thereafter restrict its behavior to blind forwarding of packets, in both directions, until the tunnel is closed. Tunnels are commonly used to create an end-to-end virtual connection, through one or more proxies, which can then be secured using TLS (Transport Layer Security, [RFC5246]).*

In Node.js, these request objects come from the `'connect'` event on the HTTP server.

When this module is used on a HTTP `CONNECT` request, the request is considered "finished" immediately, **due to limitations in the Node.js interface**. This means if the `CONNECT` request contains a request entity, the request will be considered "finished" even before it has been read.

There is no such thing as a response object to a `CONNECT` request in Node.js, so there is no support for one.

HTTP Upgrade request

The meaning of the `Upgrade` header from RFC 7230, section 6.1:

> *The "Upgrade" header field is intended to provide a simple mechanism for transitioning from HTTP/1.1 to some other protocol on the same connection.*

In Node.js, these request objects come from the `'upgrade'` event on the HTTP server.

When this module is used on a HTTP request with an `Upgrade` header, the request is considered "finished" immediately, **due to limitations in the Node.js interface**. This means if the `Upgrade` request contains a request entity, the request will be considered "finished" even before it has been read.

There is no such thing as a response object to a `Upgrade` request in Node.js, so there is no support for one.

Example

The following code ensures that file descriptors are always closed once the response finishes.

```
var destroy = require('destroy')
var fs = require('fs')
var http = require('http')
var onFinished = require('on-finished')

http.createServer(function onRequest
        (req, res) {
  var stream =
        fs.createReadStream('package.json')
  stream.pipe(res)
  onFinished(res, function () {
    destroy(stream)
  })
})
```