# send

[NPM Version](https://npmjs.org/package/send) [NPM Downloads](https://npmjs.org/package/send) [Linux Build](https://github.com/pillarjs/send/actions/workflows/ci.yml) [Windows Build](https://ci.appveyor.com/project/dougwilson/send) [Test Coverage](https://coveralls.io/r/pillarjs/send?branch=master)

Send is a library for streaming files from the file system as a http response supporting partial responses (Ranges), conditional-GET negotiation (If-Match, If-Unmodified-Since, If-None-Match, If-Modified-Since), high test coverage, and granular events which may be leveraged to take appropriate actions in your application or framework.

Looking to serve up entire folders mapped to URLs? Try [serve-static](https://www.npmjs.org/package/serve-static).

## Installation

This is a [Node.js](https://nodejs.org/en/) module available through the [npm registry](https://www.npmjs.com/). Installation is done using the [npm install command](https://docs.npmjs.com/getting-started/installing-npm-packages-locally):

$ npm install send

## API

var send = require('send')

### send(req, path, [options])

Create a new SendStream for the given path to send to a res. The req is the Node.js HTTP request and the path is a urlencoded path to send (urlencoded, not the actual file-system path).

#### Options

##### acceptRanges

Enable or disable accepting ranged requests, defaults to true. Disabling this will not send Accept-Ranges and ignore the contents of the Range request header.

##### cacheControl

Enable or disable setting Cache-Control response header, defaults to true. Disabling this will ignore the immutable and maxAge options.

##### dotfiles

Set how "dotfiles" are treated when encountered. A dotfile is a file or directory that begins with a dot ("."). Note this check is done on the path itself without checking if the path actually exists on the disk. If root is specified, only the dotfiles above the root are checked (i.e. the root itself can be within a dotfile when when set to "deny").

* 'allow' No special treatment for dotfiles.
* 'deny' Send a 403 for any request for a dotfile.
* 'ignore' Pretend like the dotfile does not exist and 404.

The default value is *similar* to 'ignore', with the exception that this default will not ignore the files within a directory that begins with a dot, for backward-compatibility.

##### end

Byte offset at which the stream ends, defaults to the length of the file minus 1. The end is inclusive in the stream, meaning end: 3 will include the 4th byte in the stream.

##### etag

Enable or disable etag generation, defaults to true.

##### extensions

If a given file doesn't exist, try appending one of the given extensions, in the given order. By default, this is disabled (set to false). An example value that will serve extension-less HTML files: ['html', 'htm']. This is skipped if the requested file already has an extension.

##### immutable

Enable or disable the immutable directive in the Cache-Control response header, defaults to false. If set to true, the maxAge option should also be specified to enable caching. The immutable directive will prevent supported clients from making conditional requests during the life of the maxAge option to check if the file has changed.

##### index

By default send supports "index.html" files, to disable this set false or to supply a new index pass a string or an array in preferred order.

##### lastModified

Enable or disable Last-Modified header, defaults to true. Uses the file system's last modified value.

##### maxAge

Provide a max-age in milliseconds for http caching, defaults to 0. This can also be a string accepted by the [ms](https://www.npmjs.org/package/ms#readme) module.

##### root

Serve files relative to path.

##### start

Byte offset at which the stream starts, defaults to 0. The start is inclusive, meaning start: 2 will include the 3rd byte in the stream.

#### Events

The SendStream is an event emitter and will emit the following events:

* error an error occurred (err)
* directory a directory was requested (res, path)
* file a file was requested (path, stat)
* headers the headers are about to be set on a file (res, path, stat)
* stream file streaming has started (stream)
* end streaming has completed

#### .pipe

The pipe method is used to pipe the response into the Node.js HTTP response object, typically send(req, path, options).pipe(res).

### .mime

The mime export is the global instance of of the [mime npm module](https://www.npmjs.com/package/mime).

This is used to configure the MIME types that are associated with file extensions as well as other options for how to resolve the MIME type of a file (like the default type to use for an unknown file extension).

## Error-handling

By default when no error listeners are present an automatic response will be made, otherwise you have full control over the response, aka you may show a 5xx page etc.

## Caching

It does *not* perform internal caching, you should use a reverse proxy cache such as Varnish for this, or those fancy things called CDNs. If your application is small enough that it would benefit from single-node memory caching, it's small enough that it does not need caching at all ;).

## Debugging

To enable debug() instrumentation output export **DEBUG**:

$ DEBUG=send node app

## Running tests

$ npm install

$ npm test

## Examples

### Serve a specific file

This simple example will send a specific file to all requests.

var http = require('http')

var send = require('send')

var server = http.createServer(function onRequest (req, res) {

send(req, '/path/to/index.html')

.pipe(res)

})

server.listen(3000)

### Serve all files from a directory

This simple example will just serve up all the files in a given directory as the top-level. For example, a request GET /foo.txt will send back /www/public/foo.txt.

var http = require('http')

var parseUrl = require('parseurl')

var send = require('send')

var server = http.createServer(function onRequest (req, res) {

send(req, parseUrl(req).pathname, { root: '/www/public' })

.pipe(res)

})

server.listen(3000)

### Custom file types

var http = require('http')

var parseUrl = require('parseurl')

var send = require('send')

// Default unknown types to text/plain

send.mime.default\_type = 'text/plain'

// Add a custom type

send.mime.define({

'application/x-my-type': ['x-mt', 'x-mtt']

})

var server = http.createServer(function onRequest (req, res) {

send(req, parseUrl(req).pathname, { root: '/www/public' })

.pipe(res)

})

server.listen(3000)

### Custom directory index view

This is a example of serving up a structure of directories with a custom function to render a listing of a directory.

var http = require('http')

var fs = require('fs')

var parseUrl = require('parseurl')

var send = require('send')

// Transfer arbitrary files from within /www/example.com/public/\*

// with a custom handler for directory listing

var server = http.createServer(function onRequest (req, res) {

send(req, parseUrl(req).pathname, { index: false, root: '/www/public' })

.once('directory', directory)

.pipe(res)

})

server.listen(3000)

// Custom directory handler

function directory (res, path) {

var stream = this

// redirect to trailing slash for consistent url

if (!stream.hasTrailingSlash()) {

return stream.redirect(path)

}

// get directory list

fs.readdir(path, function onReaddir (err, list) {

if (err) return stream.error(err)

// render an index for the directory

res.setHeader('Content-Type', 'text/plain; charset=UTF-8')

res.end(list.join('\n') + '\n')

})

}

### Serving from a root directory with custom error-handling

var http = require('http')

var parseUrl = require('parseurl')

var send = require('send')

var server = http.createServer(function onRequest (req, res) {

// your custom error-handling logic:

function error (err) {

res.statusCode = err.status || 500

res.end(err.message)

}

// your custom headers

function headers (res, path, stat) {

// serve all files for download

res.setHeader('Content-Disposition', 'attachment')

}

// your custom directory handling logic:

function redirect () {

res.statusCode = 301

res.setHeader('Location', req.url + '/')

res.end('Redirecting to ' + req.url + '/')

}

// transfer arbitrary files from within

// /www/example.com/public/\*

send(req, parseUrl(req).pathname, { root: '/www/public' })

.on('error', error)

.on('directory', redirect)

.on('headers', headers)

.pipe(res)

})

server.listen(3000)

## License

[MIT](http://license)