# raw-body

[NPM Version](https://npmjs.org/package/raw-body) [NPM Downloads](https://npmjs.org/package/raw-body) [Node.js Version](https://nodejs.org/en/download/) [Build status](https://github.com/jshttp/stream-utils/raw-body?query=workflow%3Aci) [Test coverage](https://coveralls.io/r/stream-utils/raw-body?branch=master)

Gets the entire buffer of a stream either as a Buffer or a string. Validates the stream's length against an expected length and maximum limit. Ideal for parsing request bodies.

## Install

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 raw-body

### TypeScript

This module includes a [TypeScript](https://www.typescriptlang.org/) declaration file to enable auto complete in compatible editors and type information for TypeScript projects. This module depends on the Node.js types, so install @types/node:

$ npm install @types/node

## API

var getRawBody = require('raw-body')

### getRawBody(stream, [options], [callback])

**Returns a promise if no callback specified and global Promise exists.**

Options:

* length - The length of the stream. If the contents of the stream do not add up to this length, an 400 error code is returned.
* limit - The byte limit of the body. This is the number of bytes or any string format supported by [bytes](https://www.npmjs.com/package/bytes), for example 1000, '500kb' or '3mb'. If the body ends up being larger than this limit, a 413 error code is returned.
* encoding - The encoding to use to decode the body into a string. By default, a Buffer instance will be returned when no encoding is specified. Most likely, you want utf-8, so setting encoding to true will decode as utf-8. You can use any type of encoding supported by [iconv-lite](https://www.npmjs.org/package/iconv-lite#readme).

You can also pass a string in place of options to just specify the encoding.

If an error occurs, the stream will be paused, everything unpiped, and you are responsible for correctly disposing the stream. For HTTP requests, you may need to finish consuming the stream if you want to keep the socket open for future requests. For streams that use file descriptors, you should stream.destroy() or stream.close() to prevent leaks.

## Errors

This module creates errors depending on the error condition during reading. The error may be an error from the underlying Node.js implementation, but is otherwise an error created by this module, which has the following attributes:

* limit - the limit in bytes
* length and expected - the expected length of the stream
* received - the received bytes
* encoding - the invalid encoding
* status and statusCode - the corresponding status code for the error
* type - the error type

### Types

The errors from this module have a type property which allows for the programmatic determination of the type of error returned.

#### encoding.unsupported

This error will occur when the encoding option is specified, but the value does not map to an encoding supported by the [iconv-lite](https://www.npmjs.org/package/iconv-lite#readme) module.

#### entity.too.large

This error will occur when the limit option is specified, but the stream has an entity that is larger.

#### request.aborted

This error will occur when the request stream is aborted by the client before reading the body has finished.

#### request.size.invalid

This error will occur when the length option is specified, but the stream has emitted more bytes.

#### stream.encoding.set

This error will occur when the given stream has an encoding set on it, making it a decoded stream. The stream should not have an encoding set and is expected to emit Buffer objects.

#### stream.not.readable

This error will occur when the given stream is not readable.

## Examples

### Simple Express example

var contentType = require('content-type')

var express = require('express')

var getRawBody = require('raw-body')

var app = express()

app.use(function (req, res, next) {

getRawBody(req, {

length: req.headers['content-length'],

limit: '1mb',

encoding: contentType.parse(req).parameters.charset

}, function (err, string) {

if (err) return next(err)

req.text = string

next()

})

})

// now access req.text

### Simple Koa example

var contentType = require('content-type')

var getRawBody = require('raw-body')

var koa = require('koa')

var app = koa()

app.use(function \* (next) {

this.text = yield getRawBody(this.req, {

length: this.req.headers['content-length'],

limit: '1mb',

encoding: contentType.parse(this.req).parameters.charset

})

yield next

})

// now access this.text

### Using as a promise

To use this library as a promise, simply omit the callback and a promise is returned, provided that a global Promise is defined.

var getRawBody = require('raw-body')

var http = require('http')

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

getRawBody(req)

.then(function (buf) {

res.statusCode = 200

res.end(buf.length + ' bytes submitted')

})

.catch(function (err) {

res.statusCode = 500

res.end(err.message)

})

})

server.listen(3000)

### Using with TypeScript

import \* as getRawBody from 'raw-body';

import \* as http from 'http';

const server = http.createServer((req, res) => {

getRawBody(req)

.then((buf) => {

res.statusCode = 200;

res.end(buf.length + ' bytes submitted');

})

.catch((err) => {

res.statusCode = err.statusCode;

res.end(err.message);

});

});

server.listen(3000);

## License

[MIT](http://license)