# body-parser

[NPM Version](https://npmjs.org/package/body-parser) [NPM Downloads](https://npmjs.org/package/body-parser) [Build Status](https://github.com/expressjs/body-parser/actions/workflows/ci.yml) [Test Coverage](https://coveralls.io/r/expressjs/body-parser?branch=master)

Node.js body parsing middleware.

Parse incoming request bodies in a middleware before your handlers, available under the req.body property.

**Note** As req.body's shape is based on user-controlled input, all properties and values in this object are untrusted and should be validated before trusting. For example, req.body.foo.toString() may fail in multiple ways, for example the foo property may not be there or may not be a string, and toString may not be a function and instead a string or other user input.

[Learn about the anatomy of an HTTP transaction in Node.js](https://nodejs.org/en/docs/guides/anatomy-of-an-http-transaction/).

*This does not handle multipart bodies*, due to their complex and typically large nature. For multipart bodies, you may be interested in the following modules:

* [busboy](https://www.npmjs.org/package/busboy#readme) and [connect-busboy](https://www.npmjs.org/package/connect-busboy#readme)
* [multiparty](https://www.npmjs.org/package/multiparty#readme) and [connect-multiparty](https://www.npmjs.org/package/connect-multiparty#readme)
* [formidable](https://www.npmjs.org/package/formidable#readme)
* [multer](https://www.npmjs.org/package/multer#readme)

This module provides the following parsers:

* [JSON body parser](#bodyparserjsonoptions)
* [Raw body parser](#bodyparserrawoptions)
* [Text body parser](#bodyparsertextoptions)
* [URL-encoded form body parser](#bodyparserurlencodedoptions)

Other body parsers you might be interested in:

* [body](https://www.npmjs.org/package/body#readme)
* [co-body](https://www.npmjs.org/package/co-body#readme)

## Installation

$ npm install body-parser

## API

var bodyParser = require('body-parser')

The bodyParser object exposes various factories to create middlewares. All middlewares will populate the req.body property with the parsed body when the Content-Type request header matches the type option, or an empty object ({}) if there was no body to parse, the Content-Type was not matched, or an error occurred.

The various errors returned by this module are described in the [errors section](#errors).

### bodyParser.json([options])

Returns middleware that only parses json and only looks at requests where the Content-Type header matches the type option. This parser accepts any Unicode encoding of the body and supports automatic inflation of gzip and deflate encodings.

A new body object containing the parsed data is populated on the request object after the middleware (i.e. req.body).

#### Options

The json function takes an optional options object that may contain any of the following keys:

##### inflate

When set to true, then deflated (compressed) bodies will be inflated; when false, deflated bodies are rejected. Defaults to true.

##### limit

Controls the maximum request body size. If this is a number, then the value specifies the number of bytes; if it is a string, the value is passed to the [bytes](https://www.npmjs.com/package/bytes) library for parsing. Defaults to '100kb'.

##### reviver

The reviver option is passed directly to JSON.parse as the second argument. You can find more information on this argument [in the MDN documentation about JSON.parse](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse#Example.3A_Using_the_reviver_parameter).

##### strict

When set to true, will only accept arrays and objects; when false will accept anything JSON.parse accepts. Defaults to true.

##### type

The type option is used to determine what media type the middleware will parse. This option can be a string, array of strings, or a function. If not a function, type option is passed directly to the [type-is](https://www.npmjs.org/package/type-is#readme) library and this can be an extension name (like json), a mime type (like application/json), or a mime type with a wildcard (like \*/\* or \*/json). If a function, the type option is called as fn(req) and the request is parsed if it returns a truthy value. Defaults to application/json.

##### verify

The verify option, if supplied, is called as verify(req, res, buf, encoding), where buf is a Buffer of the raw request body and encoding is the encoding of the request. The parsing can be aborted by throwing an error.

### bodyParser.raw([options])

Returns middleware that parses all bodies as a Buffer and only looks at requests where the Content-Type header matches the type option. This parser supports automatic inflation of gzip and deflate encodings.

A new body object containing the parsed data is populated on the request object after the middleware (i.e. req.body). This will be a Buffer object of the body.

#### Options

The raw function takes an optional options object that may contain any of the following keys:

##### inflate

When set to true, then deflated (compressed) bodies will be inflated; when false, deflated bodies are rejected. Defaults to true.

##### limit

Controls the maximum request body size. If this is a number, then the value specifies the number of bytes; if it is a string, the value is passed to the [bytes](https://www.npmjs.com/package/bytes) library for parsing. Defaults to '100kb'.

##### type

The type option is used to determine what media type the middleware will parse. This option can be a string, array of strings, or a function. If not a function, type option is passed directly to the [type-is](https://www.npmjs.org/package/type-is#readme) library and this can be an extension name (like bin), a mime type (like application/octet-stream), or a mime type with a wildcard (like \*/\* or application/\*). If a function, the type option is called as fn(req) and the request is parsed if it returns a truthy value. Defaults to application/octet-stream.

##### verify

The verify option, if supplied, is called as verify(req, res, buf, encoding), where buf is a Buffer of the raw request body and encoding is the encoding of the request. The parsing can be aborted by throwing an error.

### bodyParser.text([options])

Returns middleware that parses all bodies as a string and only looks at requests where the Content-Type header matches the type option. This parser supports automatic inflation of gzip and deflate encodings.

A new body string containing the parsed data is populated on the request object after the middleware (i.e. req.body). This will be a string of the body.

#### Options

The text function takes an optional options object that may contain any of the following keys:

##### defaultCharset

Specify the default character set for the text content if the charset is not specified in the Content-Type header of the request. Defaults to utf-8.

##### inflate

When set to true, then deflated (compressed) bodies will be inflated; when false, deflated bodies are rejected. Defaults to true.

##### limit

Controls the maximum request body size. If this is a number, then the value specifies the number of bytes; if it is a string, the value is passed to the [bytes](https://www.npmjs.com/package/bytes) library for parsing. Defaults to '100kb'.

##### type

The type option is used to determine what media type the middleware will parse. This option can be a string, array of strings, or a function. If not a function, type option is passed directly to the [type-is](https://www.npmjs.org/package/type-is#readme) library and this can be an extension name (like txt), a mime type (like text/plain), or a mime type with a wildcard (like \*/\* or text/\*). If a function, the type option is called as fn(req) and the request is parsed if it returns a truthy value. Defaults to text/plain.

##### verify

The verify option, if supplied, is called as verify(req, res, buf, encoding), where buf is a Buffer of the raw request body and encoding is the encoding of the request. The parsing can be aborted by throwing an error.

### bodyParser.urlencoded([options])

Returns middleware that only parses urlencoded bodies and only looks at requests where the Content-Type header matches the type option. This parser accepts only UTF-8 encoding of the body and supports automatic inflation of gzip and deflate encodings.

A new body object containing the parsed data is populated on the request object after the middleware (i.e. req.body). This object will contain key-value pairs, where the value can be a string or array (when extended is false), or any type (when extended is true).

#### Options

The urlencoded function takes an optional options object that may contain any of the following keys:

##### extended

The extended option allows to choose between parsing the URL-encoded data with the querystring library (when false) or the qs library (when true). The "extended" syntax allows for rich objects and arrays to be encoded into the URL-encoded format, allowing for a JSON-like experience with URL-encoded. For more information, please [see the qs library](https://www.npmjs.org/package/qs#readme).

Defaults to true, but using the default has been deprecated. Please research into the difference between qs and querystring and choose the appropriate setting.

##### inflate

When set to true, then deflated (compressed) bodies will be inflated; when false, deflated bodies are rejected. Defaults to true.

##### limit

Controls the maximum request body size. If this is a number, then the value specifies the number of bytes; if it is a string, the value is passed to the [bytes](https://www.npmjs.com/package/bytes) library for parsing. Defaults to '100kb'.

##### parameterLimit

The parameterLimit option controls the maximum number of parameters that are allowed in the URL-encoded data. If a request contains more parameters than this value, a 413 will be returned to the client. Defaults to 1000.

##### type

The type option is used to determine what media type the middleware will parse. This option can be a string, array of strings, or a function. If not a function, type option is passed directly to the [type-is](https://www.npmjs.org/package/type-is#readme) library and this can be an extension name (like urlencoded), a mime type (like application/x-www-form-urlencoded), or a mime type with a wildcard (like \*/x-www-form-urlencoded). If a function, the type option is called as fn(req) and the request is parsed if it returns a truthy value. Defaults to application/x-www-form-urlencoded.

##### verify

The verify option, if supplied, is called as verify(req, res, buf, encoding), where buf is a Buffer of the raw request body and encoding is the encoding of the request. The parsing can be aborted by throwing an error.

## Errors

The middlewares provided by this module create errors using the [http-errors module](https://www.npmjs.com/package/http-errors). The errors will typically have a status/statusCode property that contains the suggested HTTP response code, an expose property to determine if the message property should be displayed to the client, a type property to determine the type of error without matching against the message, and a body property containing the read body, if available.

The following are the common errors created, though any error can come through for various reasons.

### content encoding unsupported

This error will occur when the request had a Content-Encoding header that contained an encoding but the "inflation" option was set to false. The status property is set to 415, the type property is set to 'encoding.unsupported', and the charset property will be set to the encoding that is unsupported.

### entity parse failed

This error will occur when the request contained an entity that could not be parsed by the middleware. The status property is set to 400, the type property is set to 'entity.parse.failed', and the body property is set to the entity value that failed parsing.

### entity verify failed

This error will occur when the request contained an entity that could not be failed verification by the defined verify option. The status property is set to 403, the type property is set to 'entity.verify.failed', and the body property is set to the entity value that failed verification.

### request aborted

This error will occur when the request is aborted by the client before reading the body has finished. The received property will be set to the number of bytes received before the request was aborted and the expected property is set to the number of expected bytes. The status property is set to 400 and type property is set to 'request.aborted'.

### request entity too large

This error will occur when the request body's size is larger than the "limit" option. The limit property will be set to the byte limit and the length property will be set to the request body's length. The status property is set to 413 and the type property is set to 'entity.too.large'.

### request size did not match content length

This error will occur when the request's length did not match the length from the Content-Length header. This typically occurs when the request is malformed, typically when the Content-Length header was calculated based on characters instead of bytes. The status property is set to 400 and the type property is set to 'request.size.invalid'.

### stream encoding should not be set

This error will occur when something called the req.setEncoding method prior to this middleware. This module operates directly on bytes only and you cannot call req.setEncoding when using this module. The status property is set to 500 and the type property is set to 'stream.encoding.set'.

### stream is not readable

This error will occur when the request is no longer readable when this middleware attempts to read it. This typically means something other than a middleware from this module read the request body already and the middleware was also configured to read the same request. The status property is set to 500 and the type property is set to 'stream.not.readable'.

### too many parameters

This error will occur when the content of the request exceeds the configured parameterLimit for the urlencoded parser. The status property is set to 413 and the type property is set to 'parameters.too.many'.

### unsupported charset "BOGUS"

This error will occur when the request had a charset parameter in the Content-Type header, but the iconv-lite module does not support it OR the parser does not support it. The charset is contained in the message as well as in the charset property. The status property is set to 415, the type property is set to 'charset.unsupported', and the charset property is set to the charset that is unsupported.

### unsupported content encoding "bogus"

This error will occur when the request had a Content-Encoding header that contained an unsupported encoding. The encoding is contained in the message as well as in the encoding property. The status property is set to 415, the type property is set to 'encoding.unsupported', and the encoding property is set to the encoding that is unsupported.

## Examples

### Express/Connect top-level generic

This example demonstrates adding a generic JSON and URL-encoded parser as a top-level middleware, which will parse the bodies of all incoming requests. This is the simplest setup.

var express = require('express')

var bodyParser = require('body-parser')

var app = express()

// parse application/x-www-form-urlencoded

app.use(bodyParser.urlencoded({ extended: false }))

// parse application/json

app.use(bodyParser.json())

app.use(function (req, res) {

res.setHeader('Content-Type', 'text/plain')

res.write('you posted:\n')

res.end(JSON.stringify(req.body, null, 2))

})

### Express route-specific

This example demonstrates adding body parsers specifically to the routes that need them. In general, this is the most recommended way to use body-parser with Express.

var express = require('express')

var bodyParser = require('body-parser')

var app = express()

// create application/json parser

var jsonParser = bodyParser.json()

// create application/x-www-form-urlencoded parser

var urlencodedParser = bodyParser.urlencoded({ extended: false })

// POST /login gets urlencoded bodies

app.post('/login', urlencodedParser, function (req, res) {

res.send('welcome, ' + req.body.username)

})

// POST /api/users gets JSON bodies

app.post('/api/users', jsonParser, function (req, res) {

// create user in req.body

})

### Change accepted type for parsers

All the parsers accept a type option which allows you to change the Content-Type that the middleware will parse.

var express = require('express')

var bodyParser = require('body-parser')

var app = express()

// parse various different custom JSON types as JSON

app.use(bodyParser.json({ type: 'application/\*+json' }))

// parse some custom thing into a Buffer

app.use(bodyParser.raw({ type: 'application/vnd.custom-type' }))

// parse an HTML body into a string

app.use(bodyParser.text({ type: 'text/html' }))

## License

[MIT](http://license)