## Pure JS character encoding conversion [Build Status](https://travis-ci.org/ashtuchkin/iconv-lite)

* Doesn't need native code compilation. Works on Windows and in sandboxed environments like [Cloud9](http://c9.io).
* Used in popular projects like [Express.js (body\_parser)](https://github.com/expressjs/body-parser), [Grunt](http://gruntjs.com/), [Nodemailer](http://www.nodemailer.com/), [Yeoman](http://yeoman.io/) and others.
* Faster than [node-iconv](https://github.com/bnoordhuis/node-iconv) (see below for performance comparison).
* Intuitive encode/decode API
* Streaming support for Node v0.10+
* [Deprecated] Can extend Node.js primitives (buffers, streams) to support all iconv-lite encodings.
* In-browser usage via [Browserify](https://github.com/substack/node-browserify) (~180k gzip compressed with Buffer shim included).
* Typescript [type definition file](https://github.com/ashtuchkin/iconv-lite/blob/master/lib/index.d.ts) included.
* React Native is supported (need to explicitly npm install two more modules: buffer and stream).
* License: MIT.

[NPM Stats](https://npmjs.org/packages/iconv-lite/)

## Usage

### Basic API

var iconv = require('iconv-lite');

// Convert from an encoded buffer to js string.

str = iconv.decode(Buffer.from([0x68, 0x65, 0x6c, 0x6c, 0x6f]), 'win1251');

// Convert from js string to an encoded buffer.

buf = iconv.encode("Sample input string", 'win1251');

// Check if encoding is supported

iconv.encodingExists("us-ascii")

### Streaming API (Node v0.10+)

// Decode stream (from binary stream to js strings)

http.createServer(function(req, res) {

var converterStream = iconv.decodeStream('win1251');

req.pipe(converterStream);

converterStream.on('data', function(str) {

console.log(str); // Do something with decoded strings, chunk-by-chunk.

});

});

// Convert encoding streaming example

fs.createReadStream('file-in-win1251.txt')

.pipe(iconv.decodeStream('win1251'))

.pipe(iconv.encodeStream('ucs2'))

.pipe(fs.createWriteStream('file-in-ucs2.txt'));

// Sugar: all encode/decode streams have .collect(cb) method to accumulate data.

http.createServer(function(req, res) {

req.pipe(iconv.decodeStream('win1251')).collect(function(err, body) {

assert(typeof body == 'string');

console.log(body); // full request body string

});

});

### [Deprecated] Extend Node.js own encodings

NOTE: This doesn't work on latest Node versions. See [details](https://github.com/ashtuchkin/iconv-lite/wiki/Node-v4-compatibility).

// After this call all Node basic primitives will understand iconv-lite encodings.

iconv.extendNodeEncodings();

// Examples:

buf = new Buffer(str, 'win1251');

buf.write(str, 'gbk');

str = buf.toString('latin1');

assert(Buffer.isEncoding('iso-8859-15'));

Buffer.byteLength(str, 'us-ascii');

http.createServer(function(req, res) {

req.setEncoding('big5');

req.collect(function(err, body) {

console.log(body);

});

});

fs.createReadStream("file.txt", "shift\_jis");

// External modules are also supported (if they use Node primitives, which they probably do).

request = require('request');

request({

url: "http://github.com/",

encoding: "cp932"

});

// To remove extensions

iconv.undoExtendNodeEncodings();

## Supported encodings

* All node.js native encodings: utf8, ucs2 / utf16-le, ascii, binary, base64, hex.
* Additional unicode encodings: utf16, utf16-be, utf-7, utf-7-imap.
* All widespread singlebyte encodings: Windows 125x family, ISO-8859 family, IBM/DOS codepages, Macintosh family, KOI8 family, all others supported by iconv library. Aliases like 'latin1', 'us-ascii' also supported.
* All widespread multibyte encodings: CP932, CP936, CP949, CP950, GB2312, GBK, GB18030, Big5, Shift\_JIS, EUC-JP.

See [all supported encodings on wiki](https://github.com/ashtuchkin/iconv-lite/wiki/Supported-Encodings).

Most singlebyte encodings are generated automatically from [node-iconv](https://github.com/bnoordhuis/node-iconv). Thank you Ben Noordhuis and libiconv authors!

Multibyte encodings are generated from [Unicode.org mappings](http://www.unicode.org/Public/MAPPINGS/) and [WHATWG Encoding Standard mappings](http://encoding.spec.whatwg.org/). Thank you, respective authors!

## Encoding/decoding speed

Comparison with node-iconv module (1000x256kb, on MacBook Pro, Core i5/2.6 GHz, Node v0.12.0). Note: your results may vary, so please always check on your hardware.

operation iconv@2.1.4 iconv-lite@0.4.7

----------------------------------------------------------

encode('win1251') ~96 Mb/s ~320 Mb/s

decode('win1251') ~95 Mb/s ~246 Mb/s

## BOM handling

* Decoding: BOM is stripped by default, unless overridden by passing stripBOM: false in options (f.ex. iconv.decode(buf, enc, {stripBOM: false})). A callback might also be given as a stripBOM parameter - it'll be called if BOM character was actually found.
* If you want to detect UTF-8 BOM when decoding other encodings, use [node-autodetect-decoder-stream](https://github.com/danielgindi/node-autodetect-decoder-stream) module.
* Encoding: No BOM added, unless overridden by addBOM: true option.

## UTF-16 Encodings

This library supports UTF-16LE, UTF-16BE and UTF-16 encodings. First two are straightforward, but UTF-16 is trying to be smart about endianness in the following ways:

* Decoding: uses BOM and 'spaces heuristic' to determine input endianness. Default is UTF-16LE, but can be overridden with defaultEncoding: 'utf-16be' option. Strips BOM unless stripBOM: false.
* Encoding: uses UTF-16LE and writes BOM by default. Use addBOM: false to override.

## Other notes

When decoding, be sure to supply a Buffer to decode() method, otherwise [bad things usually happen](https://github.com/ashtuchkin/iconv-lite/wiki/Use-Buffers-when-decoding).  
Untranslatable characters are set to � or ?. No transliteration is currently supported.  
Node versions 0.10.31 and 0.11.13 are buggy, don't use them (see #65, #77).

## Testing

$ git clone git@github.com:ashtuchkin/iconv-lite.git

$ cd iconv-lite

$ npm install

$ npm test

$ # To view performance:

$ node test/performance.js

$ # To view test coverage:

$ npm run coverage

$ open coverage/lcov-report/index.html