Nodemailer

Send e-mails with Node.JS

MoonMail - the easiest way to send Email Marketing Newsletters!

Nodemailer

Nodemailer 2.x

> *This documentation applies to Nodemailer version v2.7.x*

Community version

These docs apply to the unmaintained versions of Nodemailer v2 and older. For an upgraded and up to date Nodemailer v3+ documentation, see nodemailer.com homepage

Nodemailer supports:

- **Node.js 0.10+**, no ES6 shenanigans used that would break your production app
- **Unicode** to use any characters, including full emoji support ?
- **Windows** – you can install it with *npm* on Windows just like any other module, there are no compiled dependencies. Use it from Azure or from your Windows box hassle free.
- **HTML content** as well as **plain text** alternative
- **Attachments** (including attachment **streaming** for sending larger files)
- **Embedded images** in HTML
- Secure e-mail delivery using **SSL/STARTTLS**
- Different **transport methods** in addition to the built-in **SMTP support**
- Custom **Plugin support** for manipulating messages (add DKIM signatures, use markdown content instead of HTML etc.)
- Sane **XOAUTH2** login with automatic access token generation (and feedback about the updated tokens)
- Simple built-in **templating** and external template renderers through node-email-templates (*optional*)
- Proxies for SMTP connections (SOCKS, HTTP and custom connections)

Support Nodemailer development

Donate

If you want to support with Bitcoins, then my wallet address is `15Z8ADxhssKUiwP3jbbqJwA21744KMCfTM`

TL;DR Usage Example

This is a complete example to send an e-mail with plaintext and HTML body

```
var nodemailer = require('nodemailer');

// create reusable transporter object using the default SMTP transport
var transporter = nodemailer.createTransport('smtps://user%40gmail.com:pass@smtp.gmail.com');

// setup e-mail data with unicode symbols
var mailOptions = {
    from: '"Fred Foo ?" <foo@blurdybloop.com>', // sender address
    to: 'bar@blurdybloop.com, baz@blurdybloop.com', // list of receivers
    subject: 'Hello ✔', // Subject line
    text: 'Hello world ?', // plaintext body
    html: '<b>Hello world ?</b>' // html body
};

// send mail with defined transport object
transporter.sendMail(mailOptions, function(error, info){
    if(error){
        return console.log(error);
    }
    console.log('Message sent: ' + info.response);
});
```

> *Using Gmail might not work* out of the box. *See instructions for setting up Gmail SMTP here.*

Setting up

Install with npm

```
npm install nodemailer@2.7.2
```

To send e-mails you need a transporter object

```
var transporter = nodemailer.createTransport(transport[, defaults])
```

Where

- **transporter** is going to be an object that is able to send mail
- **transport** is the transport configuration object, connection url or a transport plugin instance
- **defaults** is an object that defines default values for mail options

> *You have to create the transporter object only once. If you already have a transporter object you can use it to send mail as much as you like.*

Send using SMTP

See the details about setting up a SMTP based transporter here.

Send using a transport plugin

See the details about setting up a plugin based transporter here.

If the transport plugin follows common conventions, then you can also load it dynamically with the `transport` option. This way you would not have to load the transport plugin in your code (you do need to install the transport plugin before you can use it), you only need to modify the configuration data accordingly.

```
var nodemailer = require('nodemailer');
var transporter = nodemailer.createTransport({
    transport: 'ses', // loads nodemailer-ses-transport
    accessKeyId: 'AWSACCESSKEY',
    secretAccessKey: 'AWS/Secret/key'
});
```

Sending mail

Once you have a transporter object you can send mail with it:

```
transporter.sendMail(data[, callback])
```

Where

- **data** defines the mail content (see *e-mail message fields* below)
- **callback** is an optional callback function to run once the message is delivered or it failed
  - **err** is the error object if message failed
  - **info** includes the result, the exact format depends on the transport mechanism used
    - **info.messageId** most transports *should* return the final Message-Id value used with this property
    - **info.envelope** includes the envelope object for the message
    - **info.accepted** is an array returned by SMTP transports (includes recipient addresses that were accepted by the server)
    - **info.rejected** is an array returned by SMTP transports (includes recipient addresses that were rejected by the server)
    - **info.pending** is an array returned by Direct SMTP transport. Includes recipient addresses that were temporarily rejected together with the server response
    - **response** is a string returned by SMTP transports and includes the last SMTP response from the server

> *If the message includes several recipients then the message is considered sent if at least one recipient is accepted*

If `callback` argument is not set then the method returns a Promise object. Nodemailer itself does not use Promises internally but it wraps the return into a Promise for convenience.

E-mail message fields

The following are the possible fields of an e-mail message:

Commmon fields

- **from** – The e-mail address of the sender. All e-mail addresses can be plain `'sender@server.com'` or formatted `'"Sender Name" <sender@server.com>'`, see Address formatting for details
- **to** – Comma separated list or an array of recipients e-mail addresses that will appear on the *To:* field
- **cc** – Comma separated list or an array of recipients e-mail addresses that will appear on the *Cc:* field
- **bcc** – Comma separated list or an array of recipients e-mail addresses that will appear on the *Bcc:* field
- **subject** – The subject of the e-mail
- **text** – The plaintext version of the message as an Unicode string, Buffer, Stream or an attachment-like object (`{path: '/var/data/...'}`)
- **html** – The HTML version of the message as an Unicode string, Buffer, Stream or an attachment-like object (`{path: 'http://...'}`)
- **attachments** – An array of attachment objects (see Using attachments for details). Attachments can be used for embedding images as well.

A large majority of e-mails sent look a lot like this, using only a few basic fields:

```
var mailData = {
    from: 'sender@server.com',
    to: 'receiver@sender.com',
    subject: 'Message title',
    text: 'Plaintext version of the message',
    html: 'HTML version of the message'
};
```

> *NB! All text fields (e-mail addresses, plaintext body, html body, attachment filenames) use UTF-8 as the encoding. Attachments are streamed as binary.*

More advanced fields

- **sender** – An e-mail address that will appear on the *Sender:* field (always prefer `from` if you're not sure which one to use)
- **replyTo** – An e-mail address that will appear on the *Reply-To:* field
- **inReplyTo** – The message-id this message is replying to
- **references** – Message-id list (an array or space separated string)
- **watchHtml** – Apple Watch specific HTML version of the message
- **icalEvent** – iCalendar event to use as an alternative. Same usage as with `text` or `html`. Additionally you could set `method` property (defaults to `'PUBLISH'`). See an example here
- **priority** – Sets message importance headers, either `'high'`, `'normal'` (default) or `'low'`.
- **headers** – An object or array of additional header fields (e.g. *{"X-Key-Name": "key value"}* or *[{key: "X-Key-Name", value: "val1"}, {key: "X-Key-Name", value: "val2"}]*). Read more about custom headers here
- **alternatives** – An array of alternative text contents (in addition to text and html parts) (see Using alternative content for details)
- **envelope** – optional SMTP envelope, if auto generated envelope is not suitable (see SMTP envelope for details)
- **messageId** – optional Message-Id value, random value will be generated if not set
- **date** – optional Date value, current UTC string will be used if not set
- **encoding** – identifies encoding for text/html strings (defaults to 'utf-8', other values are 'hex' and 'base64')
- **raw** – existing MIME message to use instead of generating a new one. If this value is set then you should also set the envelope object (if required) as the provided raw message is not parsed. The value could be a string, a buffer, a stream or an attachment-like object.
- **textEncoding** – force content-transfer-encoding for text values (either *quoted-printable* or *base64*). By default the best option is detected (for lots of ascii use *quoted-printable*, otherwise *base64*)
- **list** – helper for setting List-* headers (see more here)
- **disableFileAccess** if true, then does not allow to use files as content. Use it when you want to use JSON data from untrusted source as the email. If an attachment or message node tries to fetch something from a file the sending returns an error. If this field is also set in the transport options, then the value in mail data is ignored

- **disableUrlAccess** if true, then does not allow to use Urls as content. If this field is also set in the transport options, then the value in mail data is ignored

```
var mailData = {
    ...,
    headers: {
        'My-Custom-Header': 'header value'
    },
    date: new Date('2000-01-01 00:00:00')
};
```

> *NB! When using readable streams as any kind of content and sending fails then Nodemailer does not abort the already opened but not yet finished stream automatically, you need to do this yourself*

```
var htmlstream = fs.createReadStream('content.html');
transport.sendMail({html: htmlstream}, function(err){
    if(err){
        // check if htmlstream is still open and close it to clean up
    }
});
```

Available Plugins

In addition to built-in e-mail fields you can extend these by using plugins.

- **nodemailer-markdown** to use markdown for the content
- **nodemailer-dkim** to sign messages with DKIM
- **nodemailer-html-to-text** to auto generate plaintext content from html
- **nodemailer-express-handlebars** to auto generate html emails from handlebars/mustache templates
- **nodemailer-plugin-inline-base64** to convert base64 images to attachments
- **nodemailer-hashcash** to generate hashcash headers

- *add yours* (see plugin api documentation here)

Implementing plugins and transports

See plugin implementation documentation here.

License

Nodemailer is licensed under MIT license. Basically you can do whatever you want to with it

---

The Nodemailer logo was designed by Sven Kristjansen.
