

[Pro](#) [Teams](#) [Pricing](#) [Documentation](#)[Sign Up](#)[Sign In](#)[Search](#)

raw-socket

1.7.0 • [Public](#) • Published 4 years ago

[Readme](#)[Code](#) [Beta](#)[1 Dependency](#)[37 Dependents](#)[28 Versions](#)

raw-socket

This module implements raw sockets for **Node.js**.

*This module has been created primarily to facilitate implementation of the **net-ping** module.*

This module is installed using **node package manager (npm)**:

```
# This module contains C++ source code which will be compiled  
# during installation using node-gyp. A suitable build chain  
# must be configured before installation.
```

```
npm install raw-socket
```

It is loaded using the `require()` function:

```
var raw = require ("raw-socket");
```

Raw sockets can then be created, and data sent using **Node.js** `Buffer` objects:

```
var socket = raw.createSocket ({protocol: raw.Protocol.None});

socket.on ("message", function (buffer, source) {
  console.log ("received " + buffer.length + " bytes from " + source);
});

socket.send (buffer, 0, buffer.length, "1.1.1.1", function (error, bytes
  if (error)
    console.log (error.toString ());
});
```

Network Protocol Support

The raw sockets exposed by this module support IPv4 and IPv6.

Raw sockets are created using the operating systems `socket()` function, and the socket type `SOCK_RAW` specified.

Raw Socket Behaviour

Raw sockets behave in different ways depending on operating system and version, and may support different socket options.

Some operating system versions may restrict the use of raw sockets to privileged users. If this is the case an exception will be thrown on socket creation using a message similar to `Operation not permitted` (this message is likely to be different depending on operating system version).

For MAC OS X platforms, when raw socket creation fails, this module will re-attempt to create a socket using the `SOCK_DGRAM` socket type for when the protocol specified is `IPPROTO_ICMP` before throwing an exception. This interface on the MAC OS X platform provides non-privileged users access to the ICMP protocol without requiring root-level access. More information on this subject can be found in the MAC OS X [documentation](#).

The appropriate operating system documentation should be consulted to understand how raw sockets will behave before attempting to use this module.

Packet Loss Under Load

Under load raw socket can experience packet loss, this may vary from system to system depending on hardware. On some systems the `SO_RCVBUF` socket option will help to alleviate packet loss.

Keeping The **Node.js** Event Loop Alive

This module uses the `libuv` library to integrate into the **Node.js** event loop - this library is also used by **Node.js**. An underlying `libuv` library `poll_handle_t` event watcher is used to monitor the underlying operating system raw socket used by a socket object.

All the while a socket object exists, and the socket's `close()` method has not been called, the raw socket will keep the **Node.js** event loop alive which will prevent a program from exiting.

This module exports four methods which a program can use to control this behaviour.

The `pauseRecv()` and `pauseSend()` methods stop the underlying `poll_handle_t` event watcher used by a socket from monitoring for readable and writeable events. While the `resumeRecv()` and `resumeSend()` methods start the underlying `poll_handle_t` event watcher used by a socket allowing it to monitor for readable and writeable events.

Each socket object also exports the `recvPaused` and `sendPaused` boolean attributes to determine the state of the underlying `poll_handle_t` event watcher used by a socket.

Socket creation can be expensive on some platforms, and the above methods offer an alternative to closing and deleting a socket to prevent it from keeping the **Node.js** event loop alive.

The **Node.js net-ping** module offers a concrete example of using these methods. Since **Node.js** offers no raw socket support this module is used to implement ICMP echo (ping) support. Once all ping requests have been processed by the **net-ping** module the `pauseRecv()` and `pauseSend()` methods are used to allow a program to exit if required.

The following example stops the underlying `poll_handle_t` event watcher used by a socket from generating writeable events, however since readable events will still be watched for the program will not exit immediately:

```
if (! socket.recvPaused)
    socket.pauseRecv ();
```

The following can be used to resume readable events:

```
if (socket.recvPaused)
    socket.resumeRecv ();
```

The following example stops the underlying `poll_handle_t` event watcher used by a socket from generating both readable and writeable events, if no other event watchers have been setup (e.g. `setTimeout()`) the program will exit.

```
if (! socket.recvPaused)
    socket.pauseRecv ();
if (! socket.sendPaused)
    socket.pauseSend ();
```

The following can be used to resume both readable and writeable events:

```
if (socket.recvPaused)
  socket.resumeRecv ();
if (socket.sendPaused)
  socket.resumeSend ();
```

When data is sent using a sockets `send()` method the `resumeSend()` method will be called if the sockets `sendPaused` attribute is `true`, however the `resumeRecv()` method will not be called regardless of whether the sockets `recvPaused` attribute is `true` or `false`.

Constants

The following sections describe constants exported and used by this module.

raw.AddressFamily

This object contains constants which can be used for the `addressFamily` option to the `createSocket()` function exposed by this module. This option specifies the IP protocol version to use when creating the raw socket.

The following constants are defined in this object:

- `IPv4` - IPv4 protocol
- `IPv6` - IPv6 protocol

raw.Protocol

This object contains constants which can be used for the `protocol` option to the `createSocket()` function exposed by this module. This option specifies the protocol number to place in the protocol field of IP headers generated by the operating system.

The following constants are defined in this object:

- `None` - protocol number 0
- `ICMP` - protocol number 1

- TCP - protocol number 6
- UDP - protocol number 17
- ICMPv6 - protocol number 58

raw.SocketLevel

This object contains constants which can be used for the `level` parameter to the `getOption()` and `setOption()` methods exposed by this module.

The following constants are defined in this object:

- SOL_SOCKET
- IPPROTO_IP
- IPPROTO_IPV6

raw.SocketOption

This object contains constants which can be used for the `option` parameter to the `getOption()` and `setOption()` methods exposed by this module.

The following constants are defined in this object:

- SO_RCVBUF
- SO_RCVTIMEO
- SO_SNDBUF
- SO_SNDTIMEO
- IP_HDRINCL
- IP_OPTIONS
- IP_TOS
- IP_TTL
- IPV6_TTL
- IPV6_UNICAST_HOPS
- IPV6_V6ONLY

The `IPV6_TTL` socket option is not known to be defined by any operating system, it is provided in convenience to be synonymous with IPv4

For Windows platforms the following constant is also defined:

- `IPV6_HDRINCL`

For Linux platforms the following constant is also defined:

- `SO_BINDTODEVICE`

Using This Module

Raw sockets are represented by an instance of the `Socket` class. This module exports the `createSocket()` function which is used to create instances of the `Socket` class.

The module also exports a number of stubs which call through to a number of functions provided by the operating system, i.e. `htonl()`.

This module also exports a function to generate protocol checksums.

`raw.createChecksum (bufferOrObject, [bufferOrObject, ...])`

The `createChecksum()` function creates and returns a 16 bit one's complement of the one's complement sum for all the data specified in one or more **Node.js** `Buffer` objects. This is useful for creating checksums for protocols such as IP, TCP, UDP and ICMP.

The `bufferOrObject` parameter can be one of two types. The first is a **Node.js** `Buffer` object. In this case a checksum is calculated from all the data it contains. The `bufferOrObject` parameter can also be an object which must contain the following attributes:

- `buffer` - A **Node.js** `Buffer` object which contains data which to generate a checksum for
- `offset` - Skip this number of bytes from the beginning of `buffer`
- `length` - Only generate a checksum for this number of bytes in `buffer` from `offset`

The second parameter type provides control over how much of the data in a **Node.js** `Buffer` object a checksum should be generated for.

When more than one parameter is passed a single checksum is calculated as if the data in all parameters were in a single buffer. This is useful for when calculating checksums for TCP and UDP for example - where a psuedo header must be created and used for checksum calculation.

In this case two buffers can be passed, the first containing the psuedo header and the second containing the real TCP packet, and the offset and length parameters used to specify the bounds of the TCP packet payload.

The following example generates a checksum for a TCP packet and its psuedo header:

```
var sum = raw.createChecksum (pseudo_header, {buffer: tcp_packet,  
    offset: 20, length: tcp_packet.length - 20});
```

Both buffers will be treated as one, i.e. as if the data at offset 20 in tcp_packet had followed all data in pseudo_header - as if they were one buffer.

raw.writeChecksum (buffer, offset, checksum)

The writeChecksum() function writes a checksum created by the raw.createChecksum() function to the **Node.js** Buffer object buffer at offsets offset and offset + 1.

The following example generates and writes a checksum at offset 2 in a **Node.js** Buffer object:

```
raw.writeChecksum (buffer, 2, raw.createChecksum (buffer));
```

raw.htonl (uint32)

The htonl() function converts a 32 bit unsigned integer from host byte order to network byte order and returns the result. This function is simply a stub through to the operating systems htonl() function.

raw.htons (uint16)

The `htons()` function converts a 16 bit unsigned integer from host byte order to network byte order and returns the result. This function is simply a stub through to the operating systems `htons()` function.

raw.ntohl (uint32)

The `ntohl()` function converts a 32 bit unsigned integer from network byte order to host byte order and returns the result. This function is simply a stub through to the operating systems `ntohl()` function.

raw.ntohs (uint16)

The `ntohs()` function converts a 16 bit unsigned integer from network byte order to host byte order and returns the result. This function is simply a stub through to the operating systems `ntohs()` function.

raw.createSocket ([options])

The `createSocket()` function instantiates and returns an instance of the `Socket` class:

```
// Default options
var options = {
  addressFamily: raw.AddressFamily.IPv4,
  protocol: raw.Protocol.None,
  bufferSize: 4096,
  generateChecksums: false,
  checksumOffset: 0
};

var socket = raw.createSocket (options);
```

The optional `options` parameter is an object, and can contain the following items:

- `addressFamily` - Either the constant `raw.AddressFamily.IPv4` or the constant `raw.AddressFamily.IPv6`, defaults to the constant `raw.AddressFamily.IPv4`
- `protocol` - Either one of the constants defined in the `raw.Protocol` object or the protocol number to use for the socket, defaults to the constant `raw.Protocol.None`
- `bufferSize` - Size, in bytes, of the sockets internal receive buffer, defaults to 4096
- `generateChecksums` - Either `true` or `false` to enable or disable the automatic checksum generation feature, defaults to `false`
- `checksumOffset` - When `generateChecksums` is `true` specifies how many bytes to index into the send buffer to write automatically generated checksums, defaults to 0

An exception will be thrown if the underlying raw socket could not be created. The error will be an instance of the `Error` class.

The `protocol` parameter, or its default value of the constant `raw.Protocol.None`, will be specified in the protocol field of each IP header.

socket.on ("close", callback)

The `close` event is emitted by the socket when the underlying raw socket is closed.

No arguments are passed to the callback.

The following example prints a message to the console when the socket is closed:

```
socket.on ("close", function () {  
    console.log ("socket closed");  
});
```

socket.on ("error", callback)

The `error` event is emitted by the socket when an error occurs sending or receiving data.

The following arguments will be passed to the `callback` function:

- `error` - An instance of the `Error` class, the exposed `message` attribute will contain a detailed error message.

The following example prints a message to the console when an error occurs, after which the socket is closed:

```
socket.on ("error", function (error) {  
    console.log (error.toString ());  
    socket.close ();  
});
```

socket.on ("message", callback)

The `message` event is emitted by the socket when data has been received.

The following arguments will be passed to the `callback` function:

- `buffer` - A **Node.js** `Buffer` object containing the data received, the buffer will be sized to fit the data received, that is the `length` attribute of buffer will specify how many bytes were received
- `address` - For IPv4 raw sockets the dotted quad formatted source IP address of the message, e.g. `192.168.1.254`, for IPv6 raw sockets the compressed formatted source IP address of the message, e.g. `fe80::a00:27ff:fe2a:3427`

The following example prints received messages in hexadecimal to the console:

```
socket.on ("message", function (buffer, address) {  
    console.log ("received " + buffer.length + " bytes from " + address  
        + ": " + buffer.toString ("hex"));  
});
```

socket.generateChecksums (generate, offset)

The `generateChecksums()` method is used to specify whether automatic checksum generation should be performed by the socket.

The `generate` parameter is either `true` or `false` to enable or disable the feature. The optional `offset` parameter specifies how many bytes to index into the send buffer when writing the generated checksum to the send buffer.

The following example enables automatic checksum generation at offset 2 resulting in checksums being written to byte 3 and 4 of the send buffer (offsets start from 0, meaning byte 1):

```
socket.generateChecksums (true, 2);
```

socket.getOption (level, option, buffer, length)

The `getOption()` method gets a socket option using the operating systems `getsockopt()` function.

The `level` parameter is one of the constants defined in the `raw.SocketLevel` object. The `option` parameter is one of the constants defined in the `raw.SocketOption` object. The `buffer` parameter is a **Node.js** `Buffer` object where the socket option value will be written. The `length` parameter specifies the size of the `buffer` parameter.

If an error occurs an exception will be thrown, the exception will be an instance of the `Error` class.

The number of bytes written into the `buffer` parameter is returned, and can differ from the amount of space available.

The following example retrieves the current value of `IP_TTL` socket option:

```
var level = raw.SocketLevel.IPPROTO_IP;
var option = raw.SocketOption.IP_TTL;

# IP_TTL is a signed integer on some platforms so a 4 byte buffer is use
var buffer = new Buffer (4);

var written = socket.getOption (level, option, buffer, buffer.length);
```

```
console.log (buffer.toString ("hex"), 0, written);
```

socket.send (buffer, offset, length, address, beforeCallback, afterCallback)

The `send()` method sends data to a remote host.

The `buffer` parameter is a **Node.js** `Buffer` object containing the data to be sent. The `length` parameter specifies how many bytes from `buffer`, beginning at `offset`, to send. For IPv4 raw sockets the `address` parameter contains the dotted quad formatted IP address of the remote host to send the data to, e.g. `192.168.1.254`, for IPv6 raw sockets the `address` parameter contains the compressed formatted IP address of the remote host to send the data to, e.g. `fe80::a00:27ff:fe2a:3427`. If provided the optional `beforeCallback` function is called right before the data is actually sent using the underlying raw socket, giving users the opportunity to perform pre-send actions such as setting a socket option, e.g. the IP header TTL. No arguments are passed to the `beforeCallback` function. The `afterCallback` function is called once the data has been sent. The following arguments will be passed to the `afterCallback` function:

- `error` - Instance of the `Error` class, or `null` if no error occurred
- `bytes` - Number of bytes sent

The following example sends a ICMP ping message to a remote host, before the request is actually sent the IP header TTL is modified, and modified again after the data has been sent:

```
// ICMP echo (ping) request, checksum should be ok
var buffer = new Buffer ([
    0x08, 0x00, 0x43, 0x52, 0x00, 0x01, 0x0a, 0x09,
    0x61, 0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68,
    0x69, 0x6a, 0x6b, 0x6c, 0x6d, 0x6e, 0x6f, 0x70,
    0x71, 0x72, 0x73, 0x74, 0x75, 0x76, 0x77, 0x61,
    0x62, 0x63, 0x64, 0x65, 0x66, 0x67, 0x68, 0x69]);
```

```
var socketLevel = raw.SocketLevel.IPPROTO_IP
var socketOption = raw.SocketOption.IP_TTL;

function beforeSend () {
    socket.setOption (socketLevel, socketOption, 1);
}

function afterSend (error, bytes) {
    if (error)
        console.log (error.toString ());
    else
        console.log ("sent " + bytes + " bytes");

    socket.setOption (socketLevel, socketOption, 1);
}

socket.send (buffer, 0, buffer.length, target, beforeSend, afterSend);
```

socket.setOption (level, option, buffer, length)

The `setOption()` method sets a socket option using the operating systems `setsockopt()` function.

The `level` parameter is one of the constants defined in the `raw.SocketLevel` object. The `option` parameter is one of the constants defined in the `raw.SocketOption` object. The `buffer` parameter is a **Node.js** `Buffer` object where the socket option value is specified. The `length` parameter specifies how much space the option value occupies in the `buffer` parameter.

If an error occurs an exception will be thrown, the exception will be an instance of the `Error` class.

The following example sets the value of `IP_TTL` socket option to `1` :

```
var level = raw.SocketLevel.IPPROTO_IP;
var option = raw.SocketOption.IP_TTL;

# IP_TTL is a signed integer on some platforms so a 4 byte buffer is use
# x86 computers use little-endian format so specify bytes reverse order
var buffer = new Buffer ([0x01, 0x00, 0x00, 0x00]);

socket.setOption (level, option, buffer, buffer.length);
```

To avoid dealing with endianness the `setOption()` method supports a three argument form which can be used for socket options requiring a 32bit unsigned integer value (for example the `IP_TTL` socket option used in the previous example). Its signature is as follows:

```
socket.setOption (level, option, value)
```

The previous example can be re-written to use this form:

```
var level = raw.SocketLevel.IPPROTO_IP;
var option = raw.SocketOption.IP_TTL;

socket.setOption (level, option, 1);
```

Example Programs

Example programs are included under the modules `example` directory.

Changes

Version 1.0.0 - 29/01/2013

- Initial release

Version 1.0.1 - 01/02/2013

- Move `SOCKET_ERRNO` define from `raw.cc` to `raw.h`
- Error in exception thrown by `SocketWrap::New` in `raw.cc` stated that two arguments were required, this should be one
- Corrections to the `README.md`
- Missing includes causes compilation error on some systems (maybe Node version dependant)

Version 1.0.2 - 02/02/2013

- Support automatic checksum generation

Version 1.1.0 - 13/02/2013

- The `net-ping` module is now implemented so update the note about it in the first section of the `README.md`
- Support IPv6
- Support the `IP_HDRINCL` socket option via the `noIpHeader` option to the `createSocket()` function and the `noIpHeader()` method exposed by the `Socket` class

Version 1.1.1 - 14/02/2013

- IP addresses not being validated

Version 1.1.2 - 15/02/2013

- Default protocol option to `createSession()` was incorrect in the `README.md`
- The `session.on("message")` example used `message` instead of `buffer` in the `README.md`

Version 1.1.3 - 04/03/2013

- `raw.Socket.onSendReady()` emit's an error when `raw.SocketWrap.send()` throws an exception when it should call the `req.callback` callback
- Added the `pauseRecv()`, `resumeRecv()`, `pauseSend()` and `resumeSend()` methods

Version 1.1.4 - 05/03/2013

- Cleanup documentation for the `pauseSend()`, `pauseRecv()`, `resumeSend()` and `resumeRecv()` methods in the README.md

Version 1.1.5 - 09/05/2013

- Reformatted lines in the README.md file inline with the rest of the file
- Removed the `noIpHeader()` method (the `setOption()` method should be used to configure the `IP_HDRINCL` socket option - and possibly `IPV6_HDRINCL` on Windows platforms), and removed the Automatic IP Header Generation section from the README.md file
- Added the `setOption()` and `getOption()` methods, and added the `SocketLevel` and `SocketOption` constants
- Tidied up the example program `ping-no-ip-header.js` (now uses the `setOption()` method to configure the `IP_HDRINCL` socket option)
- Tidied up the example program `ping6-no-ip-header.js` (now uses the `setOption()` method to configure the `IPV6_HDRINCL` socket option)
- Added the example program `get-option.js`
- Added the example program `ping-set-option-ip-ttl.js`
- Use MIT license instead of GPL

Version 1.1.6 - 18/05/2013

- Added the `beforeCallback` parameter to the `send()` method, and renamed the `callback` parameter to `afterCallback`
- Fixed a few typos in the README.md file
- Modified the example program `ping-set-option-ip-ttl.js` to use the `beforeCallback` parameter to the `send()` method
- The example program `ping6-no-ip-header.js` was not passing the correct arguments to the `setOption()` method

Version 1.1.7 - 23/06/2013

- Added the `htonl()`, `htons()`, `ntohl()`, and `ntohs()` functions, and associated example programs
- Added the `createChecksum()` function, and associated example program

Version 1.1.8 - 01/07/2013

- Added the `writeChecksum()` function
- Removed the "Automated Checksum Generation" feature - this has been replaced with the `createChecksum()` and `writeChecksum()` functions

Version 1.2.0 - 02/07/2013

- Up version number to 1.2.0 (we should have done this for 1.1.8 because it introduced some API breaking changes)

Version 1.2.1 - 15/08/2013

- Receiving Assertion `'!(handle->flags & (UV_CLOSING | UV_CLOSED))'` failed error after a number of pings - the underlying `uv_poll_t` handle was being closed twice

Version 1.2.2 - 21/09/2013

- Using `uint16_t` instead of `uint32_t` on line 87 in `src/raw.cc` for a value that is out of range
- `raw::SocketWrap::pause()` only uses the first argument
- Delete `uv_poll_t` watcher in `uv_close()` `OnClose` callback instead of in the wrapped C++ objects destructor

Version 1.3.0 - 10/07/2015

- Support Node.js 0.12.x using the Native Abstractions for Node interface
- Added export for the `SO_BINDTODEVICE` socket option for Linux platforms
- On MAC OS X platforms re-attempt to create a socket using `SOCK_DGRAM` instead of `SOCK_RAW` when `IPPROTO_ICMP` was requested by the user, this provides non-

privileged users access to the ICMP protocol on this platform

Version 1.3.1 - 10/07/2015

- Missing bracket for when compiling under the MAC OS X platform :(

Version 1.3.2 - 03/08/2015

- Add version dependency "<2.0.0" for the "nan" module to prevent build failures during installation because of breaking API changes

Version 1.3.3 - 22/09/2015

- Host repository on GitHub

Version 1.4.0 - 09/10/2015

- Support Native Abstractions for Node 2.x
- Add-on module crashes when emitting `close` events during garbage collection of a wrapped object
- Support Node.js 4.x

Version 1.5.0 - 15/05/2016

- Require nan 2.3.x to support node version 6

Version 1.5.1 - 16/11/2016

- Explicitly publish to npm using UNIX line endings

Version 1.5.2 - 11/01/2018

- Add note to README.md on how to reduce packet loss using the `SO_RCVBUF` socket option
- Address warnings for `v8::Value::ToUint32` was declared deprecated

Version 1.6.0 - 02/05/2018

- Support Node.js 10

Version 1.6.1 - 06/06/2018

- Set NoSpaceships Ltd to be the owner and maintainer

Version 1.6.2 - 07/06/2018

- Remove redundant sections from README.md

Version 1.6.3 - 03/10/2018

- Include addon `.node` file extension for node-webkit compatibility

Version 1.6.4 - 02/11/2018

- Prevent assertion failures when closing a socket and calling `pauseRecv()`

Version 1.7.0 - 12/06/2019

- Support Node.js 12 using nan 2.14

License

Copyright (c) 2018 NoSpaceships Ltd hello@nospaceships.com

Copyright (c) 2013 Stephen Vickers stephen.vickers.sv@gmail.com

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Keywords

checksum checksums htonl htons net network ntohl ntohs raw
raw-socket raw-sockets socket sockets

Install

```
> npm i raw-socket
```



Repository

 github.com/nospaceships/node-raw-socket

Homepage

 github.com/nospaceships/node-raw-socket#readme

Weekly Downloads

3,383



Version

License

1.7.0**MIT****Unpacked Size****66.2 kB****Total Files****17****Issues****8****Pull Requests****3****Last publish****4 years ago****Collaborators****NS**[>Try on RunKit](#)[🚩Report malware](#)**Support**

[Help](#)

[Advisories](#)

[Status](#)

[Contact npm](#)

Company

[About](#)

[Blog](#)

[Press](#)

Terms & Policies

[Policies](#)

[Terms of Use](#)

[Code of Conduct](#)

[Privacy](#)