Class: ClientRequest

> *Make HTTP/HTTPS requests.*

Process: Main
This class is not exported from the `'electron'` module. It is only available as a return value of other methods in the Electron API.

`ClientRequest` implements the Writable Stream interface and is therefore an EventEmitter.

`new ClientRequest(options)`

- `options` (Object | string) - If `options` is a string, it is interpreted as the request URL. If it is an object, it is expected to fully specify an HTTP request via the following properties:
  - `method` string (optional) - The HTTP request method. Defaults to the GET method.
  - `url` string (optional) - The request URL. Must be provided in the absolute form with the protocol scheme specified as http or https.
  - `session` Session (optional) - The `Session` instance with which the request is associated.
  - `partition` string (optional) - The name of the `partition` with which the request is associated. Defaults to the empty string. The `session` option supersedes `partition`. Thus if a `session` is explicitly specified, `partition` is ignored.
  - `credentials` string (optional) - Can be `include` or `omit`. Whether to send credentials with this request. If set to `include`, credentials from the session associated with the request will be used. If set to `omit`, credentials will not be sent with the request (and the `'login'` event will not be triggered in the event of a 401). This matches the behavior of the fetch option of the same name. If this option is not specified, authentication data from the session will be sent, and cookies will not be sent (unless `useSessionCookies` is set).
  - `useSessionCookies` boolean (optional) - Whether to send cookies with this request from the provided session. If `credentials` is specified, this option has no effect. Default is `false`.
  - `protocol` string (optional) - Can be `http:` or `https:`. The protocol scheme in the form 'scheme:'. Defaults to 'http:'.
  - `host` string (optional) - The server host provided as a concatenation of the hostname and the port number 'hostname:port'.
  - `hostname` string (optional) - The server host name.
  - `port` Integer (optional) - The server's listening port number.
  - `path` string (optional) - The path part of the request URL.
  - `redirect` string (optional) - Can be `follow`, `error` or `manual`. The redirect mode for this request. When mode is `error`, any redirection will be aborted. When mode is `manual` the redirection will be cancelled unless `request.followRedirect` is invoked synchronously during the `redirect` event. Defaults to `follow`.
  - `origin` string (optional) - The origin URL of the request.

`options` properties such as `protocol`, `host`, `hostname`, `port` and `path` strictly follow the Node.js model as described in the URL module.

For instance, we could have created the same request to 'github.com' as follows:

```
const request = net.request({
  method: 'GET',
  protocol: 'https:',
  hostname: 'github.com',
  port: 443,
  path: '/'
})
```

Instance Events

Event: 'response'

Returns:

- `response` IncomingMessage - An object representing the HTTP response message.

Event: 'login'

Returns:

- `authInfo` Object
  - `isProxy` boolean
  - `scheme` string
  - `host` string
  - `port` Integer
  - `realm` string
- `callback` Function
  - `username` string (optional)
  - `password` string (optional)

Emitted when an authenticating proxy is asking for user credentials.

The `callback` function is expected to be called back with user credentials:

- `username` string
- `password` string

```
request.on('login', (authInfo, callback) => {
  callback('username', 'password')
})
```

Providing empty credentials will cancel the request and report an authentication error on the response object:

```
request.on('response', (response) => {
  console.log(`STATUS: ${response.statusCode}`);
  response.on('error', (error) => {
    console.log(`ERROR: ${JSON.stringify(error)}`)
  })
})
request.on('login', (authInfo, callback) => {
  callback()
})
```

Event: 'finish'

Emitted just after the last chunk of the `request`'s data has been written into the `request` object.

Event: 'abort'

Emitted when the `request` is aborted. The `abort` event will not be fired if the `request` is already closed.

Event: 'error'

Returns:

- `error` Error - an error object providing some information about the failure.

Emitted when the `net` module fails to issue a network request. Typically when the `request` object emits an `error` event, a `close` event will subsequently follow and no response object will be provided.

Event: 'close'

Emitted as the last event in the HTTP request-response transaction. The `close` event indicates that no more events will be emitted on either the `request` or `response` objects.

Event: 'redirect'

Returns:

- `statusCode` Integer
- `method` string
- `redirectUrl` string
- `responseHeaders` Record<string, string[]>

Emitted when the server returns a redirect response (e.g. 301 Moved Permanently). Calling `request.followRedirect` will continue with the redirection. If this event is handled, `request.followRedirect` must be called **synchronously**, otherwise the request will be cancelled.

Instance Properties

`request.chunkedEncoding`

A `boolean` specifying whether the request will use HTTP chunked transfer encoding or not. Defaults to false. The property is readable and writable, however it can be set only before the first write operation as the HTTP headers are not yet put on the wire. Trying to set the `chunkedEncoding` property after the first write will throw an error.

Using chunked encoding is strongly recommended if you need to send a large request body as data will be streamed in small chunks instead of being internally buffered inside Electron process memory.

Instance Methods

`request.setHeader(name, value)`

- `name` string - An extra HTTP header name.
- `value` string - An extra HTTP header value.

Adds an extra HTTP header. The header name will be issued as-is without lowercasing. It can be called only before first write. Calling this method after the first write will throw an error. If the passed value is not a `string`, its `toString()` method will be called to obtain the final value.

Certain headers are restricted from being set by apps. These headers are listed below. More information on restricted headers can be found in Chromium's header utils.

- `Content-Length`
- `Host`
- `Trailer` or `Te`
- `Upgrade`
- `Cookie2`
- `Keep-Alive`
- `Transfer-Encoding`

Additionally, setting the `Connection` header to the value `upgrade` is also disallowed.

`request.getHeader(name)`

- `name` string - Specify an extra header name.

Returns `string` - The value of a previously set extra header name.

`request.removeHeader(name)`

- `name` string - Specify an extra header name.

Removes a previously set extra header name. This method can be called only before first write. Trying to call it after the first write will throw an error.

`request.write(chunk[, encoding][, callback])`

- `chunk` (string | Buffer) - A chunk of the request body's data. If it is a string, it is converted into a Buffer using the specified encoding.
- `encoding` string (optional) - Used to convert string chunks into Buffer objects. Defaults to 'utf-8'.
- `callback` Function (optional) - Called after the write operation ends.

`callback` is essentially a dummy function introduced in the purpose of keeping similarity with the Node.js API. It is called asynchronously in the next tick after `chunk` content have been delivered to the Chromium networking layer. Contrary to the Node.js implementation, it is not guaranteed that `chunk` content have been flushed on the wire before `callback` is called.

Adds a chunk of data to the request body. The first write operation may cause the request headers to be issued on the wire. After the first write operation, it is not allowed to add or remove a custom header.

`request.end([chunk][, encoding][, callback])`

- `chunk` (string | Buffer) (optional)
- `encoding` string (optional)
- `callback` Function (optional)

Sends the last chunk of the request data. Subsequent write or end operations will not be allowed. The `finish` event is emitted just after the end operation.

`request.abort()`

Cancels an ongoing HTTP transaction. If the request has already emitted the `close` event, the abort operation will have no effect. Otherwise an ongoing event will emit `abort` and `close` events. Additionally, if there is an ongoing response object,it will emit the `aborted` event.

`request.followRedirect()`

Continues any pending redirection. Can only be called during a `'redirect'` event.

`request.getUploadProgress()`

Returns `Object`:

- `active` boolean - Whether the request is currently active. If this is false no other properties will be set
- `started` boolean - Whether the upload has started. If this is false both `current` and `total` will be set to 0.
- `current` Integer - The number of bytes that have been uploaded so far
- `total` Integer - The number of bytes that will be uploaded this request

You can use this method in conjunction with `POST` requests to get the progress of a file upload or other data transfer.