Errors

Applications running in Node.js will generally experience four categories of errors:

- Standard JavaScript errors such as {EvalError}, {SyntaxError}, {RangeError}, {ReferenceError}, {TypeError}, and {URIError}.
- System errors triggered by underlying operating system constraints such as attempting to open a file that does not exist or attempting to send data over a closed socket.
- User-specified errors triggered by application code.
- `AssertionError`s are a special class of error that can be triggered when Node.js detects an exceptional logic violation that should never occur. These are raised typically by the `assert` module.

All JavaScript and system errors raised by Node.js inherit from, or are instances of, the standard JavaScript {Error} class and are guaranteed to provide *at least* the properties available on that class.

Error propagation and interception

Node.js supports several mechanisms for propagating and handling errors that occur while an application is running. How these errors are reported and handled depends entirely on the type of `Error` and the style of the API that is called.

All JavaScript errors are handled as exceptions that *immediately* generate and throw an error using the standard JavaScript `throw` mechanism. These are handled using the `try...catch` construct provided by the JavaScript language.

```
// Throws with a ReferenceError because z is not defined.
try {
  const m = 1;
  const n = m + z;
} catch (err) {
  // Handle the error here.
}
```

Any use of the JavaScript `throw` mechanism will raise an exception that *must* be handled using `try...catch` or the Node.js process will exit immediately.

With few exceptions, *Synchronous* APIs (any blocking method that does not accept a `callback` function, such as `fs.readFileSync`), will use `throw` to report errors.

Errors that occur within *Asynchronous APIs* may be reported in multiple ways:

- Most asynchronous methods that accept a `callback` function will accept an `Error` object passed as the first argument to that function. If that first argument is not `null` and is an instance of `Error`, then an error occurred that should be handled.

```
const fs = require('fs');
fs.readFile('a file that does not exist', (err, data) => {
  if (err) {
    console.error('There was an error reading the file!', err);
    return;
  }
  // Otherwise handle the data
});
```

- When an asynchronous method is called on an object that is an `EventEmitter`, errors can be routed to that object's `'error'` event.

```
const net = require('net');
const connection = net.connect('localhost');

// Adding an 'error' event handler to a stream:
connection.on('error', (err) => {
  // If the connection is reset by the server, or if it can't
  // connect at all, or on any sort of error encountered by
  // the connection, the error will be sent here.
  console.error(err);
});

connection.pipe(process.stdout);
```

- A handful of typically asynchronous methods in the Node.js API may still use the `throw` mechanism to raise exceptions that must be handled using `try...catch`. There is no comprehensive list of such methods; please refer to the documentation of each method to determine the appropriate error handling mechanism required.

The use of the `'error'` event mechanism is most common for stream-based and event emitter-based APIs, which themselves represent a series of asynchronous operations over time (as opposed to a single operation that may pass or fail).

For *all* `EventEmitter` objects, if an `'error'` event handler is not provided, the error will be thrown, causing the Node.js process to report an uncaught exception and crash unless either: The `domain` module is used appropriately or a handler has been registered for the `'uncaughtException'` event.

```
const EventEmitter = require('events');
const ee = new EventEmitter();

setImmediate(() => {
  // This will crash the process because no 'error' event
  // handler has been added.
  ee.emit('error', new Error('This will crash'));
});
```

Errors generated in this way *cannot* be intercepted using `try...catch` as they are thrown *after* the calling code has already exited.

Developers must refer to the documentation for each method to determine exactly how errors raised by those methods are propagated.

Error-first callbacks

Most asynchronous methods exposed by the Node.js core API follow an idiomatic pattern referred to as an *error-first callback*. With this pattern, a callback function is passed to the method as an argument. When the operation either completes or an error is raised, the callback function is called with the `Error` object (if any) passed as the first argument. If no error was raised, the first argument will be passed as `null`.

```
const fs = require('fs');

function errorFirstCallback(err, data) {
  if (err) {
    console.error('There was an error', err);
    return;
  }
  console.log(data);
}

fs.readFile('/some/file/that/does-not-exist', errorFirstCallback);
fs.readFile('/some/file/that/does-exist', errorFirstCallback);
```

The JavaScript `try...catch` mechanism **cannot** be used to intercept errors generated by asynchronous APIs. A common mistake for beginners is to try to use `throw` inside an error-first callback:

```
// THIS WILL NOT WORK:
const fs = require('fs');

try {
  fs.readFile('/some/file/that/does-not-exist', (err, data) => {
    // Mistaken assumption: throwing here...
    if (err) {
      throw err;
    }
  });
} catch (err) {
  // This will not catch the throw!
  console.error(err);
}
```

This will not work because the callback function passed to `fs.readFile()` is

called asynchronously. By the time the callback has been called, the surrounding code, including the `try...catch` block, will have already exited. Throwing an error inside the callback **can crash the Node.js process** in most cases. If domains are enabled, or a handler has been registered with `process.on('uncaughtException')`, such errors can be intercepted.

Class: `Error`

A generic JavaScript {Error} object that does not denote any specific circumstance of why the error occurred. `Error` objects capture a "stack trace" detailing the point in the code at which the `Error` was instantiated, and may provide a text description of the error.

All errors generated by Node.js, including all system and JavaScript errors, will either be instances of, or inherit from, the `Error` class.

`new Error(message)`

- `message` {string}

Creates a new `Error` object and sets the `error.message` property to the provided text message. If an object is passed as `message`, the text message is generated by calling `message.toString()`. The `error.stack` property will represent the point in the code at which `new Error()` was called. Stack traces are dependent on V8's stack trace API. Stack traces extend only to either (a) the beginning of *synchronous code execution*, or (b) the number of frames given by the property `Error.stackTraceLimit`, whichever is smaller.

`Error.captureStackTrace(targetObject[, constructorOpt])`

- `targetObject` {Object}
- `constructorOpt` {Function}

Creates a `.stack` property on `targetObject`, which when accessed returns a string representing the location in the code at which `Error.captureStackTrace()` was called.

```
const myObject = {};
Error.captureStackTrace(myObject);
myObject.stack;  // Similar to `new Error().stack`
```

The first line of the trace will be prefixed with `${myObject.name}: ${myObject.message}`.

The optional `constructorOpt` argument accepts a function. If given, all frames above `constructorOpt`, including `constructorOpt`, will be omitted from the generated stack trace.

The `constructorOpt` argument is useful for hiding implementation details of error generation from the user. For instance:

```
function MyError() {
  Error.captureStackTrace(this, MyError);
}

// Without passing MyError to captureStackTrace, the MyError
// frame would show up in the .stack property. By passing
// the constructor, we omit that frame, and retain all frames below it.
new MyError().stack;
```

`Error.stackTraceLimit`

- {number}

The `Error.stackTraceLimit` property specifies the number of stack frames collected by a stack trace (whether generated by `new Error().stack` or `Error.captureStackTrace(obj)`).

The default value is `10` but may be set to any valid JavaScript number. Changes will affect any stack trace captured *after* the value has been changed.

If set to a non-number value, or set to a negative number, stack traces will not capture any frames.

`error.code`

- {string}

The `error.code` property is a string label that identifies the kind of error. `error.code` is the most stable way to identify an error. It will only change between major versions of Node.js. In contrast, `error.message` strings may change between any versions of Node.js. See Node.js error codes for details about specific codes.

`error.message`

- {string}

The `error.message` property is the string description of the error as set by calling `new Error(message)`. The `message` passed to the constructor will also appear in the first line of the stack trace of the `Error`, however changing this property after the `Error` object is created *may not* change the first line of the stack trace (for example, when `error.stack` is read before this property is changed).

```
const err = new Error('The message');
console.error(err.message);
// Prints: The message
```

`error.stack`

- {string}

The `error.stack` property is a string describing the point in the code at which the `Error` was instantiated.

```
Error: Things keep happening!
   at /home/gbusey/file.js:525:2
   at Frobnicator.refrobulate (/home/gbusey/business-logic.js:424:21)
   at Actor.<anonymous> (/home/gbusey/actors.js:400:8)
   at increaseSynergy (/home/gbusey/actors.js:701:6)
```

The first line is formatted as `<error class name>: <error message>`, and is followed by a series of stack frames (each line beginning with "at"). Each frame describes a call site within the code that lead to the error being generated. V8 attempts to display a name for each function (by variable name, function name, or object method name), but occasionally it will not be able to find a suitable name. If V8 cannot determine a name for the function, only location information will be displayed for that frame. Otherwise, the determined function name will be displayed with location information appended in parentheses.

Frames are only generated for JavaScript functions. If, for example, execution synchronously passes through a C++ addon function called `cheetahify` which itself calls a JavaScript function, the frame representing the `cheetahify` call will not be present in the stack traces:

```
const cheetahify = require('./native-binding.node');

function makeFaster() {
  // `cheetahify()` *synchronously* calls speedy.
  cheetahify(function speedy() {
    throw new Error('oh no!');
  });
}

makeFaster();
// will throw:
//   /home/gbusey/file.js:6
//        throw new Error('oh no!');
//              ^
//   Error: oh no!
//        at speedy (/home/gbusey/file.js:6:11)
//        at makeFaster (/home/gbusey/file.js:5:3)
//        at Object.<anonymous> (/home/gbusey/file.js:10:1)
//        at Module._compile (module.js:456:26)
//        at Object.Module._extensions..js (module.js:474:10)
//        at Module.load (module.js:356:32)
```

```
//       at Function.Module._load (module.js:312:12)
//       at Function.Module.runMain (module.js:497:10)
//       at startup (node.js:119:16)
//       at node.js:906:3
```

The location information will be one of:

- `native`, if the frame represents a call internal to V8 (as in `[].forEach`).
- `plain-filename.js:line:column`, if the frame represents a call internal to Node.js.
- `/absolute/path/to/file.js:line:column`, if the frame represents a call in a user program (using CommonJS module system), or its dependencies.
- `<transport-protocol>:///url/to/module/file.mjs:line:column`, if the frame represents a call in a user program (using ES module system), or its dependencies.

The string representing the stack trace is lazily generated when the `error.stack` property is **accessed**.

The number of frames captured by the stack trace is bounded by the smaller of `Error.stackTraceLimit` or the number of available frames on the current event loop tick.

Class: `AssertionError`

- Extends: {errors.Error}

Indicates the failure of an assertion. For details, see `Class: assert.AssertionError`.

Class: `RangeError`

- Extends: {errors.Error}

Indicates that a provided argument was not within the set or range of acceptable values for a function; whether that is a numeric range, or outside the set of options for a given function parameter.

```
require('net').connect(-1);
// Throws "RangeError: "port" option should be >= 0 and < 65536: -1"
```

Node.js will generate and throw `RangeError` instances *immediately* as a form of argument validation.

Class: `ReferenceError`

- Extends: {errors.Error}

Indicates that an attempt is being made to access a variable that is not defined. Such errors commonly indicate typos in code, or an otherwise broken program.

While client code may generate and propagate these errors, in practice, only V8 will do so.

```
doesNotExist;
// Throws ReferenceError, doesNotExist is not a variable in this program.
```

Unless an application is dynamically generating and running code, `ReferenceError` instances indicate a bug in the code or its dependencies.

Class: `SyntaxError`

- Extends: {errors.Error}

Indicates that a program is not valid JavaScript. These errors may only be generated and propagated as a result of code evaluation. Code evaluation may happen as a result of `eval`, `Function`, `require`, or vm. These errors are almost always indicative of a broken program.

```
try {
  require('vm').runInThisContext('binary ! isNotOk');
} catch (err) {
  // 'err' will be a SyntaxError.
}
```

`SyntaxError` instances are unrecoverable in the context that created them – they may only be caught by other contexts.

Class: `SystemError`

- Extends: {errors.Error}

Node.js generates system errors when exceptions occur within its runtime environment. These usually occur when an application violates an operating system constraint. For example, a system error will occur if an application attempts to read a file that does not exist.

- `address` {string} If present, the address to which a network connection failed
- `code` {string} The string error code
- `dest` {string} If present, the file path destination when reporting a file system error
- `errno` {number} The system-provided error number
- `info` {Object} If present, extra details about the error condition
- `message` {string} A system-provided human-readable description of the error
- `path` {string} If present, the file path when reporting a file system error
- `port` {number} If present, the network connection port that is not available
- `syscall` {string} The name of the system call that triggered the error

`error.address`

- {string}

If present, `error.address` is a string describing the address to which a network connection failed.

`error.code`

- {string}

The `error.code` property is a string representing the error code.

`error.dest`

- {string}

If present, `error.dest` is the file path destination when reporting a file system error.

`error.errno`

- {number}

The `error.errno` property is a negative number which corresponds to the error code defined in `libuv Error handling`.

On Windows the error number provided by the system will be normalized by libuv.

To get the string representation of the error code, use `util.getSystemErrorName(error.errno)`.

`error.info`

- {Object}

If present, `error.info` is an object with details about the error condition.

`error.message`

- {string}

`error.message` is a system-provided human-readable description of the error.

`error.path`

- {string}

If present, `error.path` is a string containing a relevant invalid pathname.

`error.port`

- {number}

If present, `error.port` is the network connection port that is not available.

`error.syscall`

- {string}

The `error.syscall` property is a string describing the syscall that failed.

Common system errors

This is a list of system errors commonly-encountered when writing a Node.js program. For a comprehensive list, see the `errno`(3) man page.

- `EACCES` (Permission denied): An attempt was made to access a file in a way forbidden by its file access permissions.
- `EADDRINUSE` (Address already in use): An attempt to bind a server (`net`, `http`, or `https`) to a local address failed due to another server on the local system already occupying that address.
- `ECONNREFUSED` (Connection refused): No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host.
- `ECONNRESET` (Connection reset by peer): A connection was forcibly closed by a peer. This normally results from a loss of the connection on the remote socket due to a timeout or reboot. Commonly encountered via the `http` and `net` modules.
- `EEXIST` (File exists): An existing file was the target of an operation that required that the target not exist.
- `EISDIR` (Is a directory): An operation expected a file, but the given pathname was a directory.
- `EMFILE` (Too many open files in system): Maximum number of file descriptors allowable on the system has been reached, and requests for another descriptor cannot be fulfilled until at least one has been closed. This is encountered when opening many files at once in parallel, especially on systems (in particular, macOS) where there is a low file descriptor limit for processes. To remedy a low limit, run `ulimit -n 2048` in the same shell that will run the Node.js process.
- `ENOENT` (No such file or directory): Commonly raised by `fs` operations to indicate that a component of the specified pathname does not exist. No entity (file or directory) could be found by the given path.

- `ENOTDIR` (Not a directory): A component of the given pathname existed, but was not a directory as expected. Commonly raised by `fs.readdir`.
- `ENOTEMPTY` (Directory not empty): A directory with entries was the target of an operation that requires an empty directory, usually `fs.unlink`.
- `ENOTFOUND` (DNS lookup failed): Indicates a DNS failure of either `EAI_NODATA` or `EAI_NONAME`. This is not a standard POSIX error.
- `EPERM` (Operation not permitted): An attempt was made to perform an operation that requires elevated privileges.
- `EPIPE` (Broken pipe): A write on a pipe, socket, or FIFO for which there is no process to read the data. Commonly encountered at the `net` and `http` layers, indicative that the remote side of the stream being written to has been closed.
- `ETIMEDOUT` (Operation timed out): A connect or send request failed because the connected party did not properly respond after a period of time. Usually encountered by `http` or `net`. Often a sign that a `socket.end()` was not properly called.

Class: `TypeError`

- Extends {errors.Error}

Indicates that a provided argument is not an allowable type. For example, passing a function to a parameter which expects a string would be a `TypeError`.

```
require('url').parse(() => { });
// Throws TypeError, since it expected a string.
```

Node.js will generate and throw `TypeError` instances *immediately* as a form of argument validation.

Exceptions vs. errors

A JavaScript exception is a value that is thrown as a result of an invalid operation or as the target of a `throw` statement. While it is not required that these values are instances of `Error` or classes which inherit from `Error`, all exceptions thrown by Node.js or the JavaScript runtime *will* be instances of `Error`.

Some exceptions are *unrecoverable* at the JavaScript layer. Such exceptions will *always* cause the Node.js process to crash. Examples include `assert()` checks or `abort()` calls in the C++ layer.

OpenSSL errors

Errors originating in `crypto` or `tls` are of class `Error`, and in addition to the standard `.code` and `.message` properties, may have some additional OpenSSL-specific properties.

`error.opensslErrorStack`

An array of errors that can give context to where in the OpenSSL library an error originates from.

`error.function`

The OpenSSL function the error originates in.

`error.library`

The OpenSSL library the error originates in.

`error.reason`

A human-readable string describing the reason for the error.

Node.js error codes

`ABORT_ERR`

Used when an operation has been aborted (typically using an `AbortController`).

APIs *not* using `AbortSignal`s typically do not raise an error with this code.

This code does not use the regular `ERR_*` convention Node.js errors use in order to be compatible with the web platform's `AbortError`.

`ERR_AMBIGUOUS_ARGUMENT`

A function argument is being used in a way that suggests that the function signature may be misunderstood. This is thrown by the `assert` module when the `message` parameter in `assert.throws(block, message)` matches the error message thrown by `block` because that usage suggests that the user believes `message` is the expected message rather than the message the `AssertionError` will display if `block` does not throw.

`ERR_ARG_NOT_ITERABLE`

An iterable argument (i.e. a value that works with `for...of` loops) was required, but not provided to a Node.js API.

`ERR_ASSERTION`

A special type of error that can be triggered whenever Node.js detects an exceptional logic violation that should never occur. These are raised typically by the `assert` module.

`ERR_ASYNC_CALLBACK`

An attempt was made to register something that is not a function as an `AsyncHooks` callback.

`ERR_ASYNC_TYPE`

The type of an asynchronous resource was invalid. Users are also able to define their own types if using the public embedder API.

`ERR_BROTLI_COMPRESSION_FAILED`

Data passed to a Brotli stream was not successfully compressed.

`ERR_BROTLI_INVALID_PARAM`

An invalid parameter key was passed during construction of a Brotli stream.

`ERR_BUFFER_CONTEXT_NOT_AVAILABLE`

An attempt was made to create a Node.js `Buffer` instance from addon or embedder code, while in a JS engine Context that is not associated with a Node.js instance. The data passed to the `Buffer` method will have been released by the time the method returns.

When encountering this error, a possible alternative to creating a `Buffer` instance is to create a normal `Uint8Array`, which only differs in the prototype of the resulting object. `Uint8Array`s are generally accepted in all Node.js core APIs where `Buffer`s are; they are available in all Contexts.

`ERR_BUFFER_OUT_OF_BOUNDS`

An operation outside the bounds of a `Buffer` was attempted.

`ERR_BUFFER_TOO_LARGE`

An attempt has been made to create a `Buffer` larger than the maximum allowed size.

`ERR_CANNOT_WATCH_SIGINT`

Node.js was unable to watch for the `SIGINT` signal.

`ERR_CHILD_CLOSED_BEFORE_REPLY`

A child process was closed before the parent received a reply.

`ERR_CHILD_PROCESS_IPC_REQUIRED`

Used when a child process is being forked without specifying an IPC channel.

`ERR_CHILD_PROCESS_STDIO_MAXBUFFER`

Used when the main process is trying to read data from the child process's STDERR/STDOUT, and the data's length is longer than the `maxBuffer` option.

`ERR_CLOSED_MESSAGE_PORT`

There was an attempt to use a `MessagePort` instance in a closed state, usually after `.close()` has been called.

`ERR_CONSOLE_WRITABLE_STREAM`

`Console` was instantiated without `stdout` stream, or `Console` has a non-writable `stdout` or `stderr` stream.

`ERR_CONSTRUCT_CALL_INVALID`

A class constructor was called that is not callable.

`ERR_CONSTRUCT_CALL_REQUIRED`

A constructor for a class was called without `new`.

`ERR_CONTEXT_NOT_INITIALIZED`

The vm context passed into the API is not yet initialized. This could happen when an error occurs (and is caught) during the creation of the context, for example, when the allocation fails or the maximum call stack size is reached when the context is created.

`ERR_CRYPTO_CUSTOM_ENGINE_NOT_SUPPORTED`

A client certificate engine was requested that is not supported by the version of OpenSSL being used.

`ERR_CRYPTO_ECDH_INVALID_FORMAT`

An invalid value for the `format` argument was passed to the `crypto.ECDH()` class `getPublicKey()` method.

`ERR_CRYPTO_ECDH_INVALID_PUBLIC_KEY`

An invalid value for the `key` argument has been passed to the `crypto.ECDH()` class `computeSecret()` method. It means that the public key lies outside of the elliptic curve.

`ERR_CRYPTO_ENGINE_UNKNOWN`

An invalid crypto engine identifier was passed to `require('crypto').setEngine()`.

`ERR_CRYPTO_FIPS_FORCED`

The `--force-fips` command-line argument was used but there was an attempt to enable or disable FIPS mode in the `crypto` module.

`ERR_CRYPTO_FIPS_UNAVAILABLE`

An attempt was made to enable or disable FIPS mode, but FIPS mode was not available.

`ERR_CRYPTO_HASH_FINALIZED`

`hash.digest()` was called multiple times. The `hash.digest()` method must be called no more than one time per instance of a `Hash` object.

`ERR_CRYPTO_HASH_UPDATE_FAILED`

`hash.update()` failed for any reason. This should rarely, if ever, happen.

`ERR_CRYPTO_INCOMPATIBLE_KEY`

The given crypto keys are incompatible with the attempted operation.

`ERR_CRYPTO_INCOMPATIBLE_KEY_OPTIONS`

The selected public or private key encoding is incompatible with other options.

`ERR_CRYPTO_INITIALIZATION_FAILED`

Initialization of the crypto subsystem failed.

`ERR_CRYPTO_INVALID_AUTH_TAG`

An invalid authentication tag was provided.

`ERR_CRYPTO_INVALID_COUNTER`

An invalid counter was provided for a counter-mode cipher.

`ERR_CRYPTO_INVALID_CURVE`

An invalid elliptic-curve was provided.

`ERR_CRYPTO_INVALID_DIGEST`

An invalid crypto digest algorithm was specified.

`ERR_CRYPTO_INVALID_IV`

An invalid initialization vector was provided.

`ERR_CRYPTO_INVALID_JWK`

An invalid JSON Web Key was provided.

`ERR_CRYPTO_INVALID_KEY_OBJECT_TYPE`

The given crypto key object's type is invalid for the attempted operation.

`ERR_CRYPTO_INVALID_KEYLEN`

An invalid key length was provided.

`ERR_CRYPTO_INVALID_KEYPAIR`

An invalid key pair was provided.

`ERR_CRYPTO_INVALID_KEYTYPE`

An invalid key type was provided.

`ERR_CRYPTO_INVALID_MESSAGELEN`

An invalid message length was provided.

`ERR_CRYPTO_INVALID_SCRYPT_PARAMS`

Invalid scrypt algorithm parameters were provided.

`ERR_CRYPTO_INVALID_STATE`

A crypto method was used on an object that was in an invalid state. For instance, calling `cipher.getAuthTag()` before calling `cipher.final()`.

`ERR_CRYPTO_INVALID_TAG_LENGTH`

An invalid authentication tag length was provided.

`ERR_CRYPTO_JOB_INIT_FAILED`

Initialization of an asynchronous crypto operation failed.

`ERR_CRYPTO_JWK_UNSUPPORTED_CURVE`

Key's Elliptic Curve is not registered for use in the JSON Web Key Elliptic Curve Registry.

`ERR_CRYPTO_JWK_UNSUPPORTED_KEY_TYPE`

Key's Asymmetric Key Type is not registered for use in the JSON Web Key Types Registry.

`ERR_CRYPTO_OPERATION_FAILED`

A crypto operation failed for an otherwise unspecified reason.

`ERR_CRYPTO_PBKDF2_ERROR`

The PBKDF2 algorithm failed for unspecified reasons. OpenSSL does not provide more details and therefore neither does Node.js.

`ERR_CRYPTO_SCRYPT_INVALID_PARAMETER`

One or more `crypto.scrypt()` or `crypto.scryptSync()` parameters are outside their legal range.

`ERR_CRYPTO_SCRYPT_NOT_SUPPORTED`

Node.js was compiled without `scrypt` support. Not possible with the official release binaries but can happen with custom builds, including distro builds.

`ERR_CRYPTO_SIGN_KEY_REQUIRED`

A signing `key` was not provided to the `sign.sign()` method.

`ERR_CRYPTO_TIMING_SAFE_EQUAL_LENGTH`

`crypto.timingSafeEqual()` was called with `Buffer`, `TypedArray`, or `DataView` arguments of different lengths.

`ERR_CRYPTO_UNKNOWN_CIPHER`

An unknown cipher was specified.

`ERR_CRYPTO_UNKNOWN_DH_GROUP`

An unknown Diffie-Hellman group name was given. See `crypto.getDiffieHellman()` for a list of valid group names.

`ERR_CRYPTO_UNSUPPORTED_OPERATION`

An attempt to invoke an unsupported crypto operation was made.

`ERR_DEBUGGER_ERROR`

An error occurred with the debugger.

`ERR_DEBUGGER_STARTUP_ERROR`

The debugger timed out waiting for the required host/port to be free.

`ERR_DLOPEN_DISABLED`

Loading native addons has been disabled using `--no-addons`.

`ERR_DLOPEN_FAILED`

A call to `process.dlopen()` failed.

`ERR_DIR_CLOSED`

The `fs.Dir` was previously closed.

`ERR_DIR_CONCURRENT_OPERATION`

A synchronous read or close call was attempted on an `fs.Dir` which has ongoing asynchronous operations.

`ERR_DNS_SET_SERVERS_FAILED`

`c-ares` failed to set the DNS server.

`ERR_DOMAIN_CALLBACK_NOT_AVAILABLE`

The `domain` module was not usable since it could not establish the required error handling hooks, because `process.setUncaughtExceptionCaptureCallback()` had been called at an earlier point in time.

`ERR_DOMAIN_CANNOT_SET_UNCAUGHT_EXCEPTION_CAPTURE`

`process.setUncaughtExceptionCaptureCallback()` could not be called because the `domain` module has been loaded at an earlier point in time.

The stack trace is extended to include the point in time at which the `domain` module had been loaded.

`ERR_ENCODING_INVALID_ENCODED_DATA`

Data provided to `TextDecoder()` API was invalid according to the encoding provided.

`ERR_ENCODING_NOT_SUPPORTED`

Encoding provided to `TextDecoder()` API was not one of the WHATWG Supported Encodings.

`ERR_EVAL_ESM_CANNOT_PRINT`

`--print` cannot be used with ESM input.

`ERR_EVENT_RECURSION`

Thrown when an attempt is made to recursively dispatch an event on `EventTarget`.

`ERR_EXECUTION_ENVIRONMENT_NOT_AVAILABLE`

The JS execution context is not associated with a Node.js environment. This may occur when Node.js is used as an embedded library and some hooks for the JS engine are not set up properly.

`ERR_FALSY_VALUE_REJECTION`

A `Promise` that was callbackified via `util.callbackify()` was rejected with a falsy value.

`ERR_FEATURE_UNAVAILABLE_ON_PLATFORM`

Used when a feature that is not available to the current platform which is running Node.js is used.

`ERR_FS_CP_DIR_TO_NON_DIR`

An attempt was made to copy a directory to a non-directory (file, symlink, etc.) using `fs.cp()`.

`ERR_FS_CP_EEXIST`

An attempt was made to copy over a file that already existed with `fs.cp()`, with the `force` and `errorOnExist` set to `true`.

`ERR_FS_CP_EINVAL`

When using `fs.cp()`, `src` or `dest` pointed to an invalid path.

`ERR_FS_CP_FIFO_PIPE`

An attempt was made to copy a named pipe with `fs.cp()`.

`ERR_FS_CP_NON_DIR_TO_DIR`

An attempt was made to copy a non-directory (file, symlink, etc.) to a directory using `fs.cp()`.

`ERR_FS_CP_SOCKET`

An attempt was made to copy to a socket with `fs.cp()`.

`ERR_FS_CP_SYMLINK_TO_SUBDIRECTORY`

When using `fs.cp()`, a symlink in `dest` pointed to a subdirectory of `src`.

`ERR_FS_CP_UNKNOWN`

An attempt was made to copy to an unknown file type with `fs.cp()`.

`ERR_FS_EISDIR`

Path is a directory.

`ERR_FS_FILE_TOO_LARGE`

An attempt has been made to read a file whose size is larger than the maximum allowed size for a `Buffer`.

`ERR_FS_INVALID_SYMLINK_TYPE`

An invalid symlink type was passed to the `fs.symlink()` or `fs.symlinkSync()` methods.

`ERR_HTTP_HEADERS_SENT`

An attempt was made to add more headers after the headers had already been sent.

`ERR_HTTP_INVALID_HEADER_VALUE`

An invalid HTTP header value was specified.

`ERR_HTTP_INVALID_STATUS_CODE`

Status code was outside the regular status code range (100-999).

`ERR_HTTP_REQUEST_TIMEOUT`

The client has not sent the entire request within the allowed time.

`ERR_HTTP_SOCKET_ENCODING`

Changing the socket encoding is not allowed per RFC 7230 Section 3.

`ERR_HTTP_TRAILER_INVALID`

The `Trailer` header was set even though the transfer encoding does not support that.

`ERR_HTTP2_ALTSVC_INVALID_ORIGIN`

HTTP/2 ALTSVC frames require a valid origin.

`ERR_HTTP2_ALTSVC_LENGTH`

HTTP/2 ALTSVC frames are limited to a maximum of 16,382 payload bytes.

`ERR_HTTP2_CONNECT_AUTHORITY`

For HTTP/2 requests using the `CONNECT` method, the `:authority` pseudo-header is required.

`ERR_HTTP2_CONNECT_PATH`

For HTTP/2 requests using the `CONNECT` method, the `:path` pseudo-header is forbidden.

`ERR_HTTP2_CONNECT_SCHEME`

For HTTP/2 requests using the `CONNECT` method, the `:scheme` pseudo-header is forbidden.

`ERR_HTTP2_ERROR`

A non-specific HTTP/2 error has occurred.

`ERR_HTTP2_GOAWAY_SESSION`

New HTTP/2 Streams may not be opened after the `Http2Session` has received a `GOAWAY` frame from the connected peer.

`ERR_HTTP2_HEADER_SINGLE_VALUE`

Multiple values were provided for an HTTP/2 header field that was required to have only a single value.

`ERR_HTTP2_HEADERS_AFTER_RESPOND`

An additional headers was specified after an HTTP/2 response was initiated.

`ERR_HTTP2_HEADERS_SENT`

An attempt was made to send multiple response headers.

`ERR_HTTP2_INFO_STATUS_NOT_ALLOWED`

Informational HTTP status codes (`1xx`) may not be set as the response status code on HTTP/2 responses.

`ERR_HTTP2_INVALID_CONNECTION_HEADERS`

HTTP/1 connection specific headers are forbidden to be used in HTTP/2 requests and responses.

`ERR_HTTP2_INVALID_HEADER_VALUE`

An invalid HTTP/2 header value was specified.

`ERR_HTTP2_INVALID_INFO_STATUS`

An invalid HTTP informational status code has been specified. Informational status codes must be an integer between `100` and `199` (inclusive).

`ERR_HTTP2_INVALID_ORIGIN`

HTTP/2 `ORIGIN` frames require a valid origin.

`ERR_HTTP2_INVALID_PACKED_SETTINGS_LENGTH`

Input `Buffer` and `Uint8Array` instances passed to the `http2.getUnpackedSettings()` API must have a length that is a multiple of six.

`ERR_HTTP2_INVALID_PSEUDOHEADER`

Only valid HTTP/2 pseudoheaders (`:status`, `:path`, `:authority`, `:scheme`, and `:method`) may be used.

`ERR_HTTP2_INVALID_SESSION`

An action was performed on an `Http2Session` object that had already been destroyed.

`ERR_HTTP2_INVALID_SETTING_VALUE`

An invalid value has been specified for an HTTP/2 setting.

`ERR_HTTP2_INVALID_STREAM`

An operation was performed on a stream that had already been destroyed.

`ERR_HTTP2_MAX_PENDING_SETTINGS_ACK`

Whenever an HTTP/2 `SETTINGS` frame is sent to a connected peer, the peer is required to send an acknowledgment that it has received and applied the new `SETTINGS`. By default, a maximum number of unacknowledged `SETTINGS` frames may be sent at any given time. This error code is used when that limit has been reached.

`ERR_HTTP2_NESTED_PUSH`

An attempt was made to initiate a new push stream from within a push stream. Nested push streams are not permitted.

`ERR_HTTP2_NO_MEM`

Out of memory when using the `http2session.setLocalWindowSize(windowSize)` API.

`ERR_HTTP2_NO_SOCKET_MANIPULATION`

An attempt was made to directly manipulate (read, write, pause, resume, etc.) a socket attached to an `Http2Session`.

`ERR_HTTP2_ORIGIN_LENGTH`

HTTP/2 `ORIGIN` frames are limited to a length of 16382 bytes.

`ERR_HTTP2_OUT_OF_STREAMS`

The number of streams created on a single HTTP/2 session reached the maximum limit.

`ERR_HTTP2_PAYLOAD_FORBIDDEN`

A message payload was specified for an HTTP response code for which a payload is forbidden.

`ERR_HTTP2_PING_CANCEL`

An HTTP/2 ping was canceled.

`ERR_HTTP2_PING_LENGTH`

HTTP/2 ping payloads must be exactly 8 bytes in length.

`ERR_HTTP2_PSEUDOHEADER_NOT_ALLOWED`

An HTTP/2 pseudo-header has been used inappropriately. Pseudo-headers are header key names that begin with the `:` prefix.

`ERR_HTTP2_PUSH_DISABLED`

An attempt was made to create a push stream, which had been disabled by the client.

`ERR_HTTP2_SEND_FILE`

An attempt was made to use the `Http2Stream.prototype.responseWithFile()` API to send a directory.

`ERR_HTTP2_SEND_FILE_NOSEEK`

An attempt was made to use the `Http2Stream.prototype.responseWithFile()` API to send something other than a regular file, but `offset` or `length` options were provided.

`ERR_HTTP2_SESSION_ERROR`

The `Http2Session` closed with a non-zero error code.

`ERR_HTTP2_SETTINGS_CANCEL`

The `Http2Session` settings canceled.

`ERR_HTTP2_SOCKET_BOUND`

An attempt was made to connect a `Http2Session` object to a `net.Socket` or `tls.TLSSocket` that had already been bound to another `Http2Session` object.

`ERR_HTTP2_SOCKET_UNBOUND`

An attempt was made to use the `socket` property of an `Http2Session` that has already been closed.

`ERR_HTTP2_STATUS_101`

Use of the `101` Informational status code is forbidden in HTTP/2.

`ERR_HTTP2_STATUS_INVALID`

An invalid HTTP status code has been specified. Status codes must be an integer between `100` and `599` (inclusive).

`ERR_HTTP2_STREAM_CANCEL`

An `Http2Stream` was destroyed before any data was transmitted to the connected peer.

`ERR_HTTP2_STREAM_ERROR`

A non-zero error code was been specified in an `RST_STREAM` frame.

`ERR_HTTP2_STREAM_SELF_DEPENDENCY`

When setting the priority for an HTTP/2 stream, the stream may be marked as a dependency for a parent stream. This error code is used when an attempt is made to mark a stream and dependent of itself.

`ERR_HTTP2_TOO_MANY_INVALID_FRAMES`

The limit of acceptable invalid HTTP/2 protocol frames sent by the peer, as specified through the `maxSessionInvalidFrames` option, has been exceeded.

`ERR_HTTP2_TRAILERS_ALREADY_SENT`

Trailing headers have already been sent on the `Http2Stream`.

`ERR_HTTP2_TRAILERS_NOT_READY`

The `http2stream.sendTrailers()` method cannot be called until after the `'wantTrailers'` event is emitted on an `Http2Stream` object. The `'wantTrailers'` event will only be emitted if the `waitForTrailers` option is set for the `Http2Stream`.

`ERR_HTTP2_UNSUPPORTED_PROTOCOL`

`http2.connect()` was passed a URL that uses any protocol other than `http:` or `https:`.

`ERR_ILLEGAL_CONSTRUCTOR`

An attempt was made to construct an object using a non-public constructor.

`ERR_IMPORT_ASSERTION_TYPE_FAILED`

An import assertion has failed, preventing the specified module to be imported.

`ERR_IMPORT_ASSERTION_TYPE_MISSING`

An import assertion is missing, preventing the specified module to be imported.

`ERR_IMPORT_ASSERTION_TYPE_UNSUPPORTED`

An import assertion is not supported by this version of Node.js.

`ERR_INCOMPATIBLE_OPTION_PAIR`

An option pair is incompatible with each other and cannot be used at the same time.

ERR_INPUT_TYPE_NOT_ALLOWED

Stability: 1 - Experimental

The `--input-type` flag was used to attempt to execute a file. This flag can only be used with input via `--eval`, `--print` or `STDIN`.

ERR_INSPECTOR_ALREADY_ACTIVATED

While using the `inspector` module, an attempt was made to activate the inspector when it already started to listen on a port. Use `inspector.close()` before activating it on a different address.

ERR_INSPECTOR_ALREADY_CONNECTED

While using the `inspector` module, an attempt was made to connect when the inspector was already connected.

ERR_INSPECTOR_CLOSED

While using the `inspector` module, an attempt was made to use the inspector after the session had already closed.

ERR_INSPECTOR_COMMAND

An error occurred while issuing a command via the `inspector` module.

ERR_INSPECTOR_NOT_ACTIVE

The `inspector` is not active when `inspector.waitForDebugger()` is called.

ERR_INSPECTOR_NOT_AVAILABLE

The `inspector` module is not available for use.

ERR_INSPECTOR_NOT_CONNECTED

While using the `inspector` module, an attempt was made to use the inspector before it was connected.

ERR_INSPECTOR_NOT_WORKER

An API was called on the main thread that can only be used from the worker thread.

ERR_INTERNAL_ASSERTION

There was a bug in Node.js or incorrect usage of Node.js internals. To fix the error, open an issue at https://github.com/nodejs/node/issues.

`ERR_INVALID_ADDRESS_FAMILY`

The provided address family is not understood by the Node.js API.

`ERR_INVALID_ARG_TYPE`

An argument of the wrong type was passed to a Node.js API.

`ERR_INVALID_ARG_VALUE`

An invalid or unsupported value was passed for a given argument.

`ERR_INVALID_ASYNC_ID`

An invalid `asyncId` or `triggerAsyncId` was passed using `AsyncHooks`. An id less than -1 should never happen.

`ERR_INVALID_BUFFER_SIZE`

A swap was performed on a `Buffer` but its size was not compatible with the operation.

`ERR_INVALID_CHAR`

Invalid characters were detected in headers.

`ERR_INVALID_CURSOR_POS`

A cursor on a given stream cannot be moved to a specified row without a specified column.

`ERR_INVALID_FD`

A file descriptor ('fd') was not valid (e.g. it was a negative value).

`ERR_INVALID_FD_TYPE`

A file descriptor ('fd') type was not valid.

`ERR_INVALID_FILE_URL_HOST`

A Node.js API that consumes `file:` URLs (such as certain functions in the `fs` module) encountered a file URL with an incompatible host. This situation can only occur on Unix-like systems where only `localhost` or an empty host is supported.

`ERR_INVALID_FILE_URL_PATH`

A Node.js API that consumes `file:` URLs (such as certain functions in the `fs` module) encountered a file URL with an incompatible path. The exact semantics for determining whether a path can be used is platform-dependent.

`ERR_INVALID_HANDLE_TYPE`

An attempt was made to send an unsupported "handle" over an IPC communication channel to a child process. See `subprocess.send()` and `process.send()` for more information.

`ERR_INVALID_HTTP_TOKEN`

An invalid HTTP token was supplied.

`ERR_INVALID_IP_ADDRESS`

An IP address is not valid.

`ERR_INVALID_MODULE`

An attempt was made to load a module that does not exist or was otherwise not valid.

`ERR_INVALID_MODULE_SPECIFIER`

The imported module string is an invalid URL, package name, or package subpath specifier.

`ERR_INVALID_OBJECT_DEFINE_PROPERTY`

An error occurred while setting an invalid attribute on the property of an object.

`ERR_INVALID_PACKAGE_CONFIG`

An invalid `package.json` file failed parsing.

`ERR_INVALID_PACKAGE_TARGET`

The `package.json "exports"` field contains an invalid target mapping value for the attempted module resolution.

`ERR_INVALID_PERFORMANCE_MARK`

While using the Performance Timing API (`perf_hooks`), a performance mark is invalid.

ERR_INVALID_PROTOCOL

An invalid `options.protocol` was passed to `http.request()`.

ERR_INVALID_REPL_EVAL_CONFIG

Both `breakEvalOnSigint` and `eval` options were set in the `REPL` config, which is not supported.

ERR_INVALID_REPL_INPUT

The input may not be used in the `REPL`. The conditions under which this error is used are described in the `REPL` documentation.

ERR_INVALID_RETURN_PROPERTY

Thrown in case a function option does not provide a valid value for one of its returned object properties on execution.

ERR_INVALID_RETURN_PROPERTY_VALUE

Thrown in case a function option does not provide an expected value type for one of its returned object properties on execution.

ERR_INVALID_RETURN_VALUE

Thrown in case a function option does not return an expected value type on execution, such as when a function is expected to return a promise.

ERR_INVALID_STATE

Indicates that an operation cannot be completed due to an invalid state. For instance, an object may have already been destroyed, or may be performing another operation.

ERR_INVALID_SYNC_FORK_INPUT

A `Buffer`, `TypedArray`, `DataView` or `string` was provided as stdio input to an asynchronous fork. See the documentation for the `child_process` module for more information.

ERR_INVALID_THIS

A Node.js API function was called with an incompatible `this` value.

```
const urlSearchParams = new URLSearchParams('foo=bar&baz=new');

const buf = Buffer.alloc(1);
```

```
urlSearchParams.has.call(buf, 'foo');
// Throws a TypeError with code 'ERR_INVALID_THIS'
```

ERR_INVALID_TRANSFER_OBJECT

An invalid transfer object was passed to `postMessage()`.

ERR_INVALID_TUPLE

An element in the `iterable` provided to the WHATWG `URLSearchParams` constructor did not represent a `[name, value]` tuple – that is, if an element is not iterable, or does not consist of exactly two elements.

ERR_INVALID_URI

An invalid URI was passed.

ERR_INVALID_URL

An invalid URL was passed to the WHATWG `URL` constructor or the legacy `url.parse()` to be parsed. The thrown error object typically has an additional property `'input'` that contains the URL that failed to parse.

ERR_INVALID_URL_SCHEME

An attempt was made to use a URL of an incompatible scheme (protocol) for a specific purpose. It is only used in the WHATWG URL API support in the `fs` module (which only accepts URLs with `'file'` scheme), but may be used in other Node.js APIs as well in the future.

ERR_IPC_CHANNEL_CLOSED

An attempt was made to use an IPC communication channel that was already closed.

ERR_IPC_DISCONNECTED

An attempt was made to disconnect an IPC communication channel that was already disconnected. See the documentation for the `child_process` module for more information.

ERR_IPC_ONE_PIPE

An attempt was made to create a child Node.js process using more than one IPC communication channel. See the documentation for the `child_process` module for more information.

`ERR_IPC_SYNC_FORK`

An attempt was made to open an IPC communication channel with a synchronously forked Node.js process. See the documentation for the `child_process` module for more information.

`ERR_MANIFEST_ASSERT_INTEGRITY`

An attempt was made to load a resource, but the resource did not match the integrity defined by the policy manifest. See the documentation for policy manifests for more information.

`ERR_MANIFEST_DEPENDENCY_MISSING`

An attempt was made to load a resource, but the resource was not listed as a dependency from the location that attempted to load it. See the documentation for policy manifests for more information.

`ERR_MANIFEST_INTEGRITY_MISMATCH`

An attempt was made to load a policy manifest, but the manifest had multiple entries for a resource which did not match each other. Update the manifest entries to match in order to resolve this error. See the documentation for policy manifests for more information.

`ERR_MANIFEST_INVALID_RESOURCE_FIELD`

A policy manifest resource had an invalid value for one of its fields. Update the manifest entry to match in order to resolve this error. See the documentation for policy manifests for more information.

`ERR_MANIFEST_INVALID_SPECIFIER`

A policy manifest resource had an invalid value for one of its dependency mappings. Update the manifest entry to match to resolve this error. See the documentation for policy manifests for more information.

`ERR_MANIFEST_PARSE_POLICY`

An attempt was made to load a policy manifest, but the manifest was unable to be parsed. See the documentation for policy manifests for more information.

`ERR_MANIFEST_TDZ`

An attempt was made to read from a policy manifest, but the manifest initialization has not yet taken place. This is likely a bug in Node.js.

`ERR_MANIFEST_UNKNOWN_ONERROR`

A policy manifest was loaded, but had an unknown value for its "onerror" behavior. See the documentation for policy manifests for more information.

`ERR_MEMORY_ALLOCATION_FAILED`

An attempt was made to allocate memory (usually in the C++ layer) but it failed.

`ERR_MESSAGE_TARGET_CONTEXT_UNAVAILABLE`

A message posted to a `MessagePort` could not be deserialized in the target vm `Context`. Not all Node.js objects can be successfully instantiated in any context at this time, and attempting to transfer them using `postMessage()` can fail on the receiving side in that case.

`ERR_METHOD_NOT_IMPLEMENTED`

A method is required but not implemented.

`ERR_MISSING_ARGS`

A required argument of a Node.js API was not passed. This is only used for strict compliance with the API specification (which in some cases may accept `func(undefined)` but not `func()`). In most native Node.js APIs, `func(undefined)` and `func()` are treated identically, and the `ERR_INVALID_ARG_TYPE` error code may be used instead.

`ERR_MISSING_OPTION`

For APIs that accept options objects, some options might be mandatory. This code is thrown if a required option is missing.

`ERR_MISSING_PASSPHRASE`

An attempt was made to read an encrypted key without specifying a passphrase.

`ERR_MISSING_PLATFORM_FOR_WORKER`

The V8 platform used by this instance of Node.js does not support creating Workers. This is caused by lack of embedder support for Workers. In particular, this error will not occur with standard builds of Node.js.

`ERR_MISSING_TRANSFERABLE_IN_TRANSFER_LIST`

An object that needs to be explicitly listed in the `transferList` argument is in the object passed to a `postMessage()` call, but is not provided in the `transferList` for that call. Usually, this is a `MessagePort`.

In Node.js versions prior to v15.0.0, the error code being used here was `ERR_MISSING_MESSAGE_PORT_IN_TRANSFER_LIST`. However, the set of transferable object types has been expanded to cover more types than `MessagePort`.

`ERR_MODULE_NOT_FOUND`

A module file could not be resolved by the ECMAScript modules loader while attempting an `import` operation or when loading the program entry point.

`ERR_MULTIPLE_CALLBACK`

A callback was called more than once.

A callback is almost always meant to only be called once as the query can either be fulfilled or rejected but not both at the same time. The latter would be possible by calling a callback more than once.

`ERR_NAPI_CONS_FUNCTION`

While using `Node-API`, a constructor passed was not a function.

`ERR_NAPI_INVALID_DATAVIEW_ARGS`

While calling `napi_create_dataview()`, a given `offset` was outside the bounds of the dataview or `offset + length` was larger than a length of given `buffer`.

`ERR_NAPI_INVALID_TYPEDARRAY_ALIGNMENT`

While calling `napi_create_typedarray()`, the provided `offset` was not a multiple of the element size.

`ERR_NAPI_INVALID_TYPEDARRAY_LENGTH`

While calling `napi_create_typedarray()`, `(length * size_of_element) + byte_offset` was larger than the length of given `buffer`.

`ERR_NAPI_TSFN_CALL_JS`

An error occurred while invoking the JavaScript portion of the thread-safe function.

`ERR_NAPI_TSFN_GET_UNDEFINED`

An error occurred while attempting to retrieve the JavaScript `undefined` value.

`ERR_NAPI_TSFN_START_IDLE_LOOP`

On the main thread, values are removed from the queue associated with the thread-safe function in an idle loop. This error indicates that an error has occurred when attempting to start the loop.

`ERR_NAPI_TSFN_STOP_IDLE_LOOP`

Once no more items are left in the queue, the idle loop must be suspended. This error indicates that the idle loop has failed to stop.

`ERR_NO_CRYPTO`

An attempt was made to use crypto features while Node.js was not compiled with OpenSSL crypto support.

`ERR_NO_ICU`

An attempt was made to use features that require ICU, but Node.js was not compiled with ICU support.

`ERR_NON_CONTEXT_AWARE_DISABLED`

A non-context-aware native addon was loaded in a process that disallows them.

`ERR_OUT_OF_RANGE`

A given value is out of the accepted range.

`ERR_PACKAGE_IMPORT_NOT_DEFINED`

The `package.json "imports"` field does not define the given internal package specifier mapping.

`ERR_PACKAGE_PATH_NOT_EXPORTED`

The `package.json "exports"` field does not export the requested subpath. Because exports are encapsulated, private internal modules that are not exported cannot be imported through the package resolution, unless using an absolute URL.

`ERR_PERFORMANCE_INVALID_TIMESTAMP`

An invalid timestamp value was provided for a performance mark or measure.

`ERR_PERFORMANCE_MEASURE_INVALID_OPTIONS`

Invalid options were provided for a performance measure.

`ERR_PROTO_ACCESS`

Accessing `Object.prototype.__proto__` has been forbidden using `--disable-proto=throw`. `Object.getPrototypeOf` and `Object.setPrototypeOf` should be used to get and set the prototype of an object.

`ERR_REQUIRE_ESM`

Stability: 1 - Experimental

An attempt was made to `require()` an ES Module.

`ERR_SCRIPT_EXECUTION_INTERRUPTED`

Script execution was interrupted by `SIGINT` (For example, Ctrl+C was pressed.)

`ERR_SCRIPT_EXECUTION_TIMEOUT`

Script execution timed out, possibly due to bugs in the script being executed.

`ERR_SERVER_ALREADY_LISTEN`

The `server.listen()` method was called while a `net.Server` was already listening. This applies to all instances of `net.Server`, including HTTP, HTTPS, and HTTP/2 `Server` instances.

`ERR_SERVER_NOT_RUNNING`

The `server.close()` method was called when a `net.Server` was not running. This applies to all instances of `net.Server`, including HTTP, HTTPS, and HTTP/2 `Server` instances.

`ERR_SOCKET_ALREADY_BOUND`

An attempt was made to bind a socket that has already been bound.

`ERR_SOCKET_BAD_BUFFER_SIZE`

An invalid (negative) size was passed for either the `recvBufferSize` or `sendBufferSize` options in `dgram.createSocket()`.

`ERR_SOCKET_BAD_PORT`

An API function expecting a port >= 0 and < 65536 received an invalid value.

`ERR_SOCKET_BAD_TYPE`

An API function expecting a socket type (`udp4` or `udp6`) received an invalid value.

`ERR_SOCKET_BUFFER_SIZE`

While using `dgram.createSocket()`, the size of the receive or send `Buffer` could not be determined.

`ERR_SOCKET_CLOSED`

An attempt was made to operate on an already closed socket.

`ERR_SOCKET_DGRAM_IS_CONNECTED`

A `dgram.connect()` call was made on an already connected socket.

`ERR_SOCKET_DGRAM_NOT_CONNECTED`

A `dgram.disconnect()` or `dgram.remoteAddress()` call was made on a disconnected socket.

`ERR_SOCKET_DGRAM_NOT_RUNNING`

A call was made and the UDP subsystem was not running.

`ERR_SRI_PARSE`

A string was provided for a Subresource Integrity check, but was unable to be parsed. Check the format of integrity attributes by looking at the Subresource Integrity specification.

`ERR_STREAM_ALREADY_FINISHED`

A stream method was called that cannot complete because the stream was finished.

`ERR_STREAM_CANNOT_PIPE`

An attempt was made to call `stream.pipe()` on a `Writable` stream.

`ERR_STREAM_DESTROYED`

A stream method was called that cannot complete because the stream was destroyed using `stream.destroy()`.

`ERR_STREAM_NULL_VALUES`

An attempt was made to call `stream.write()` with a `null` chunk.

`ERR_STREAM_PREMATURE_CLOSE`

An error returned by `stream.finished()` and `stream.pipeline()`, when a stream or a pipeline ends non gracefully with no explicit error.

ERR_STREAM_PUSH_AFTER_EOF

An attempt was made to call `stream.push()` after a `null`(EOF) had been pushed to the stream.

ERR_STREAM_UNSHIFT_AFTER_END_EVENT

An attempt was made to call `stream.unshift()` after the `'end'` event was emitted.

ERR_STREAM_WRAP

Prevents an abort if a string decoder was set on the Socket or if the decoder is in `objectMode`.

```
const Socket = require('net').Socket;
const instance = new Socket();

instance.setEncoding('utf8');
```

ERR_STREAM_WRITE_AFTER_END

An attempt was made to call `stream.write()` after `stream.end()` has been called.

ERR_STRING_TOO_LONG

An attempt has been made to create a string longer than the maximum allowed length.

ERR_SYNTHETIC

An artificial error object used to capture the call stack for diagnostic reports.

ERR_SYSTEM_ERROR

An unspecified or non-specific system error has occurred within the Node.js process. The error object will have an `err.info` object property with additional details.

ERR_TEST_FAILURE

This error represents a failed test. Additional information about the failure is available via the `cause` property. The `failureType` property specifies what the test was doing when the failure occurred.

`ERR_TLS_CERT_ALTNAME_FORMAT`

This error is thrown by `checkServerIdentity` if a user-supplied `subjectaltname` property violates encoding rules. Certificate objects produced by Node.js itself always comply with encoding rules and will never cause this error.

`ERR_TLS_CERT_ALTNAME_INVALID`

While using TLS, the host name/IP of the peer did not match any of the `subjectAltNames` in its certificate.

`ERR_TLS_DH_PARAM_SIZE`

While using TLS, the parameter offered for the Diffie-Hellman (`DH`) key-agreement protocol is too small. By default, the key length must be greater than or equal to 1024 bits to avoid vulnerabilities, even though it is strongly recommended to use 2048 bits or larger for stronger security.

`ERR_TLS_HANDSHAKE_TIMEOUT`

A TLS/SSL handshake timed out. In this case, the server must also abort the connection.

`ERR_TLS_INVALID_CONTEXT`

The context must be a `SecureContext`.

`ERR_TLS_INVALID_PROTOCOL_METHOD`

The specified `secureProtocol` method is invalid. It is either unknown, or disabled because it is insecure.

`ERR_TLS_INVALID_PROTOCOL_VERSION`

Valid TLS protocol versions are `'TLSv1'`, `'TLSv1.1'`, or `'TLSv1.2'`.

`ERR_TLS_INVALID_STATE`

The TLS socket must be connected and securily established. Ensure the 'secure' event is emitted before continuing.

`ERR_TLS_PROTOCOL_VERSION_CONFLICT`

Attempting to set a TLS protocol `minVersion` or `maxVersion` conflicts with an attempt to set the `secureProtocol` explicitly. Use one mechanism or the other.

`ERR_TLS_PSK_SET_IDENTIY_HINT_FAILED`

Failed to set PSK identity hint. Hint may be too long.

`ERR_TLS_RENEGOTIATION_DISABLED`

An attempt was made to renegotiate TLS on a socket instance with TLS disabled.

`ERR_TLS_REQUIRED_SERVER_NAME`

While using TLS, the `server.addContext()` method was called without providing a host name in the first parameter.

`ERR_TLS_SESSION_ATTACK`

An excessive amount of TLS renegotiations is detected, which is a potential vector for denial-of-service attacks.

`ERR_TLS_SNI_FROM_SERVER`

An attempt was made to issue Server Name Indication from a TLS server-side socket, which is only valid from a client.

`ERR_TRACE_EVENTS_CATEGORY_REQUIRED`

The `trace_events.createTracing()` method requires at least one trace event category.

`ERR_TRACE_EVENTS_UNAVAILABLE`

The `trace_events` module could not be loaded because Node.js was compiled with the `--without-v8-platform` flag.

`ERR_TRANSFORM_ALREADY_TRANSFORMING`

A `Transform` stream finished while it was still transforming.

`ERR_TRANSFORM_WITH_LENGTH_0`

A `Transform` stream finished with data still in the write buffer.

`ERR_TTY_INIT_FAILED`

The initialization of a TTY failed due to a system error.

`ERR_UNAVAILABLE_DURING_EXIT`

Function was called within a `process.on('exit')` handler that shouldn't be called within `process.on('exit')` handler.

ERR_UNCAUGHT_EXCEPTION_CAPTURE_ALREADY_SET

`process.setUncaughtExceptionCaptureCallback()` was called twice, without first resetting the callback to `null`.

This error is designed to prevent accidentally overwriting a callback registered from another module.

ERR_UNESCAPED_CHARACTERS

A string that contained unescaped characters was received.

ERR_UNHANDLED_ERROR

An unhandled error occurred (for instance, when an `'error'` event is emitted by an `EventEmitter` but an `'error'` handler is not registered).

ERR_UNKNOWN_BUILTIN_MODULE

Used to identify a specific kind of internal Node.js error that should not typically be triggered by user code. Instances of this error point to an internal bug within the Node.js binary itself.

ERR_UNKNOWN_CREDENTIAL

A Unix group or user identifier that does not exist was passed.

ERR_UNKNOWN_ENCODING

An invalid or unknown encoding option was passed to an API.

ERR_UNKNOWN_FILE_EXTENSION

> Stability: 1 - Experimental

An attempt was made to load a module with an unknown or unsupported file extension.

ERR_UNKNOWN_MODULE_FORMAT

> Stability: 1 - Experimental

An attempt was made to load a module with an unknown or unsupported format.

ERR_UNKNOWN_SIGNAL

An invalid or unknown process signal was passed to an API expecting a valid signal (such as `subprocess.kill()`).

ERR_UNSUPPORTED_DIR_IMPORT

`import` a directory URL is unsupported. Instead, self-reference a package using its name and define a custom subpath in the `"exports"` field of the `package.json` file.

```
import './'; // unsupported
import './index.js'; // supported
import 'package-name'; // supported
```

ERR_UNSUPPORTED_ESM_URL_SCHEME

`import` with URL schemes other than `file` and `data` is unsupported.

ERR_VALID_PERFORMANCE_ENTRY_TYPE

While using the Performance Timing API (`perf_hooks`), no valid performance entry types are found.

ERR_VM_DYNAMIC_IMPORT_CALLBACK_MISSING

A dynamic import callback was not specified.

ERR_VM_MODULE_ALREADY_LINKED

The module attempted to be linked is not eligible for linking, because of one of the following reasons:

- It has already been linked (`linkingStatus` is `'linked'`)
- It is being linked (`linkingStatus` is `'linking'`)
- Linking has failed for this module (`linkingStatus` is `'errored'`)

ERR_VM_MODULE_CACHED_DATA_REJECTED

The `cachedData` option passed to a module constructor is invalid.

ERR_VM_MODULE_CANNOT_CREATE_CACHED_DATA

Cached data cannot be created for modules which have already been evaluated.

ERR_VM_MODULE_DIFFERENT_CONTEXT

The module being returned from the linker function is from a different context than the parent module. Linked modules must share the same context.

ERR_VM_MODULE_LINKING_ERRORED

The linker function returned a module for which linking has failed.

`ERR_VM_MODULE_LINK_FAILURE`

The module was unable to be linked due to a failure.

`ERR_VM_MODULE_NOT_MODULE`

The fulfilled value of a linking promise is not a `vm.Module` object.

`ERR_VM_MODULE_STATUS`

The current module's status does not allow for this operation. The specific meaning of the error depends on the specific function.

`ERR_WASI_ALREADY_STARTED`

The WASI instance has already started.

`ERR_WASI_NOT_STARTED`

The WASI instance has not been started.

`ERR_WORKER_INIT_FAILED`

The `Worker` initialization failed.

`ERR_WORKER_INVALID_EXEC_ARGV`

The `execArgv` option passed to the `Worker` constructor contains invalid flags.

`ERR_WORKER_NOT_RUNNING`

An operation failed because the `Worker` instance is not currently running.

`ERR_WORKER_OUT_OF_MEMORY`

The `Worker` instance terminated because it reached its memory limit.

`ERR_WORKER_PATH`

The path for the main script of a worker is neither an absolute path nor a relative path starting with `./` or `../`.

`ERR_WORKER_UNSERIALIZABLE_ERROR`

All attempts at serializing an uncaught exception from a worker thread failed.

`ERR_WORKER_UNSUPPORTED_OPERATION`

The requested functionality is not supported in worker threads.

`ERR_ZLIB_INITIALIZATION_FAILED`

Creation of a `zlib` object failed due to incorrect configuration.

`HPE_HEADER_OVERFLOW`

Too much HTTP header data was received. In order to protect against malicious or malconfigured clients, if more than 8 KB of HTTP header data is received then HTTP parsing will abort without a request or response object being created, and an `Error` with this code will be emitted.

`HPE_UNEXPECTED_CONTENT_LENGTH`

Server is sending both a `Content-Length` header and `Transfer-Encoding: chunked`.

`Transfer-Encoding: chunked` allows the server to maintain an HTTP persistent connection for dynamically generated content. In this case, the `Content-Length` HTTP header cannot be used.

Use `Content-Length` or `Transfer-Encoding: chunked`.

`MODULE_NOT_FOUND`

A module file could not be resolved by the CommonJS modules loader while attempting a `require()` operation or when loading the program entry point.

Legacy Node.js error codes

> Stability: 0 - Deprecated. These error codes are either inconsistent, or have been removed.

`ERR_CANNOT_TRANSFER_OBJECT`

The value passed to `postMessage()` contained an object that is not supported for transferring.

`ERR_CRYPTO_HASH_DIGEST_NO_UTF16`

The UTF-16 encoding was used with `hash.digest()`. While the `hash.digest()` method does allow an `encoding` argument to be passed in, causing the method to return a string rather than a `Buffer`, the UTF-16 encoding (e.g. `ucs` or `utf16le`) is not supported.

`ERR_HTTP2_FRAME_ERROR`

Used when a failure occurs sending an individual frame on the HTTP/2 session.

`ERR_HTTP2_HEADERS_OBJECT`

Used when an HTTP/2 Headers Object is expected.

`ERR_HTTP2_HEADER_REQUIRED`

Used when a required header is missing in an HTTP/2 message.

`ERR_HTTP2_INFO_HEADERS_AFTER_RESPOND`

HTTP/2 informational headers must only be sent *prior* to calling the `Http2Stream.prototype.respond()` method.

`ERR_HTTP2_STREAM_CLOSED`

Used when an action has been performed on an HTTP/2 Stream that has already been closed.

`ERR_HTTP_INVALID_CHAR`

Used when an invalid character is found in an HTTP response status message (reason phrase).

`ERR_INDEX_OUT_OF_RANGE`

A given index was out of the accepted range (e.g. negative offsets).

`ERR_INVALID_OPT_VALUE`

An invalid or unexpected value was passed in an options object.

`ERR_INVALID_OPT_VALUE_ENCODING`

An invalid or unknown file encoding was passed.

`ERR_MISSING_MESSAGE_PORT_IN_TRANSFER_LIST`

This error code was replaced by `ERR_MISSING_TRANSFERABLE_IN_TRANSFER_LIST` in Node.js v15.0.0, because it is no longer accurate as other types of transferable objects also exist now.

`ERR_NAPI_CONS_PROTOTYPE_OBJECT`

Used by the `Node-API` when `Constructor.prototype` is not an object.

`ERR_NETWORK_IMPORT_BAD_RESPONSE`

> Stability: 1 - Experimental

Response was received but was invalid when importing a module over the network.

`ERR_NETWORK_IMPORT_DISALLOWED`

> Stability: 1 - Experimental

A network module attempted to load another module that it is not allowed to load. Likely this restriction is for security reasons.

`ERR_NO_LONGER_SUPPORTED`

A Node.js API was called in an unsupported manner, such as `Buffer.write(string, encoding, offset[, length])`.

`ERR_OPERATION_FAILED`

An operation failed. This is typically used to signal the general failure of an asynchronous operation.

`ERR_OUTOFMEMORY`

Used generically to identify that an operation caused an out of memory condition.

`ERR_PARSE_HISTORY_DATA`

The `repl` module was unable to parse data from the REPL history file.

`ERR_SOCKET_CANNOT_SEND`

Data could not be sent on a socket.

`ERR_STDERR_CLOSE`

An attempt was made to close the `process.stderr` stream. By design, Node.js does not allow `stdout` or `stderr` streams to be closed by user code.

`ERR_STDOUT_CLOSE`

An attempt was made to close the `process.stdout` stream. By design, Node.js does not allow `stdout` or `stderr` streams to be closed by user code.

`ERR_STREAM_READ_NOT_IMPLEMENTED`

Used when an attempt is made to use a readable stream that has not implemented `readable._read()`.

`ERR_TLS_RENEGOTIATION_FAILED`

Used when a TLS renegotiation request has failed in a non-specific way.

`ERR_TRANSFERRING_EXTERNALIZED_SHAREDARRAYBUFFER`

A `SharedArrayBuffer` whose memory is not managed by the JavaScript engine or by Node.js was encountered during serialization. Such a `SharedArrayBuffer` cannot be serialized.

This can only happen when native addons create `SharedArrayBuffer`s in "externalized" mode, or put existing `SharedArrayBuffer` into externalized mode.

`ERR_UNKNOWN_STDIN_TYPE`

An attempt was made to launch a Node.js process with an unknown `stdin` file type. This error is usually an indication of a bug within Node.js itself, although it is possible for user code to trigger it.

`ERR_UNKNOWN_STREAM_TYPE`

An attempt was made to launch a Node.js process with an unknown `stdout` or `stderr` file type. This error is usually an indication of a bug within Node.js itself, although it is possible for user code to trigger it.

`ERR_V8BREAKITERATOR`

The V8 `BreakIterator` API was used but the full ICU data set is not installed.

`ERR_VALUE_OUT_OF_RANGE`

Used when a given value is out of the accepted range.

`ERR_VM_MODULE_NOT_LINKED`

The module must be successfully linked before instantiation.

`ERR_WORKER_UNSUPPORTED_EXTENSION`

The pathname used for the main script of a worker has an unknown file extension.

`ERR_ZLIB_BINDING_CLOSED`

Used when an attempt is made to use a `zlib` object after it has already been closed.

`ERR_CPU_USAGE`

The native call from `process.cpuUsage` could not be processed.