Node.js Core Test Common Modules

This directory contains modules used to test the Node.js implementation.

Table of Contents

Benchmark Module

The `benchmark` module is used by tests to run benchmarks.

`runBenchmark(name, env)`

- `name` <string> Name of benchmark suite to be run.
- `env` <Object> Environment variables to be applied during the run.

Common Module API

The `common` module is used by tests for consistency across repeated tasks.

`allowGlobals(...allowlist)`

- `allowlist` <Array> Array of Globals
- return <Array>

Takes `allowlist` and concats that with predefined `knownGlobals`.

`canCreateSymLink()`

- return <boolean>

Checks whether the current running process can create symlinks. On Windows, this returns `false` if the process running doesn't have privileges to create symlinks (SeCreateSymbolicLinkPrivilege). On non-Windows platforms, this always returns `true`.

`createZeroFilledFile(filename)`

Creates a 10 MB file of all null characters.

`enoughTestMem`

- <boolean>

Indicates if there is more than 1gb of total memory.

`expectsError(validator[, exact])`

- `validator` <Object> | <RegExp> | <Function> | <Error> The validator behaves identical to `assert.throws(fn, validator)`.
- `exact` <number> default = 1
- return <Function> A callback function that expects an error.

A function suitable as callback to validate callback based errors. The error is validated using `assert.throws(() => { throw error; }, validator)`. If the returned function has not been called exactly `exact` number of times when the test is complete, then the test will fail.

`expectWarning(name[, expected[, code]])`

- `name` <string> | <Object>
- `expected` <string> | <Array> | <Object>
- `code` <string>

Tests whether `name`, `expected`, and `code` are part of a raised warning.

The code is required in case the name is set to `'DeprecationWarning'`.

Examples:

```
const { expectWarning } = require('../common');

expectWarning('Warning', 'Foobar is really bad');

expectWarning('DeprecationWarning', 'Foobar is deprecated', 'DEP0XXX');

expectWarning('DeprecationWarning', [
  'Foobar is deprecated', 'DEP0XXX',
]);

expectWarning('DeprecationWarning', [
  ['Foobar is deprecated', 'DEP0XXX'],
  ['Baz is also deprecated', 'DEP0XX2'],
]);

expectWarning('DeprecationWarning', {
  DEP0XXX: 'Foobar is deprecated',
  DEP0XX2: 'Baz is also deprecated'
});

expectWarning({
  DeprecationWarning: {
    DEP0XXX: 'Foobar is deprecated',
    DEP0XX1: 'Baz is also deprecated'
  },
  Warning: [
    ['Multiple array entries are fine', 'SpecialWarningCode'],
    ['No code is also fine'],
  ],
  SingleEntry: ['This will also work', 'WarningCode'],
  SingleString: 'Single string entries without code will also work'
});
```

`getArrayBufferViews(buf)`

- `buf` <Buffer>
- return <ArrayBufferView>[]

Returns an instance of all possible `ArrayBufferView`s of the provided Buffer.

`getBufferSources(buf)`

- `buf` <Buffer>
- return <BufferSource>[]

Returns an instance of all possible `BufferSource`s of the provided Buffer, consisting of all `ArrayBufferView` and an `ArrayBuffer`.

`getCallSite(func)`

- `func` <Function>
- return <string>

Returns the file name and line number for the provided Function.

`getTTYfd()`

Attempts to get a valid TTY file descriptor. Returns `-1` if it fails.

The TTY file descriptor is assumed to be capable of being writable.

`hasCrypto`

- <boolean>

Indicates whether OpenSSL is available.

`hasFipsCrypto`

- <boolean>

Indicates that Node.js has been linked with a FIPS compatible OpenSSL library, and that FIPS as been enabled using `--enable-fips`.

To only detect if the OpenSSL library is FIPS compatible, regardless if it has been enabled or not, then `process.config.variables.openssl_is_fips` can be used to determine that situation.

`hasIntl`

- <boolean>

Indicates if internationalization is supported.

`hasIPv6`

- <boolean>

Indicates whether `IPv6` is supported on this platform.

`hasMultiLocalhost`

- <boolean>

Indicates if there are multiple localhosts available.

`inFreeBSDJail`

- <boolean>

Checks whether free BSD Jail is true or false.

`isAIX`

- <boolean>

Platform check for Advanced Interactive eXecutive (AIX).

`isAlive(pid)`

- `pid` <number>
- return <boolean>

Attempts to 'kill' `pid`

`isDumbTerminal`

- <boolean>

`isFreeBSD`

- <boolean>

Platform check for Free BSD.

`isIBMi`

- <boolean>

Platform check for IBMi.

`isLinux`

- <boolean>

Platform check for Linux.

`isLinuxPPCBE`

- <boolean>

Platform check for Linux on PowerPC.

`isOSX`

- <boolean>

Platform check for macOS.

`isSunOS`

- <boolean>

Platform check for SunOS.

`isWindows`

- <boolean>

Platform check for Windows.

`localhostIPv4`

- <string>

IP of `localhost`.

`localIPv6Hosts`

- <Array>

Array of IPV6 representations for `localhost`.

`mustCall([fn][, exact])`

- `fn` <Function> default = () => {}
- `exact` <number> default = 1
- return <Function>

Returns a function that calls `fn`. If the returned function has not been called exactly `exact` number of times when the test is complete, then the test will fail.

If `fn` is not provided, an empty function will be used.

`mustCallAtLeast([fn][, minimum])`

- `fn` <Function> default = () => {}
- `minimum` <number> default = 1
- return <Function>

Returns a function that calls `fn`. If the returned function has not been called at least `minimum` number of times when the test is complete, then the test will fail.

If `fn` is not provided, an empty function will be used.

`mustNotCall([msg])`

- `msg` <string> default = 'function should not have been called'
- return <Function>

Returns a function that triggers an `AssertionError` if it is invoked. `msg` is used as the error message for the `AssertionError`.

`mustSucceed([fn])`

- `fn` <Function> default = () => {}
- return <Function>

Returns a function that accepts arguments `(err, ...args)`. If `err` is not `undefined` or `null`, it triggers an `AssertionError`. Otherwise, it calls `fn(...args)`.

`nodeProcessAborted(exitCode, signal)`

- `exitCode` <number>
- `signal` <string>
- return <boolean>

Returns `true` if the exit code `exitCode` and/or signal name `signal` represent the exit code and/or signal name of a node process that aborted, `false` otherwise.

`opensslCli`

- <boolean>

Indicates whether 'opensslCli' is supported.

`platformTimeout(ms)`

- `ms` <number> | <bigint>
- return <number> | <bigint>

Returns a timeout value based on detected conditions. For example, a debug build may need extra time so the returned value will be larger than on a release build.

`PIPE`

- <string>

Path to the test socket.

`PORT`

- <number>

A port number for tests to use if one is needed.

`printSkipMessage(msg)`

- `msg` <string>

Logs '1..0 # Skipped:' + `msg`

`pwdCommand`

- <array> First two argument for the `spawn`/`exec` functions.

Platform normalized `pwd` command options. Usage example:

```js
const common = require('../common');
const { spawn } = require('child_process');

spawn(...common.pwdCommand, { stdio: ['pipe'] });
```

`requireNoPackageJSONAbove([dir])`

- `dir` <string> default = ___dirname

Throws an `AssertionError` if a `package.json` file exists in any ancestor directory above `dir`. Such files may interfere with proper test functionality.

`runWithInvalidFD(func)`

- `func` <Function>

Runs `func` with an invalid file descriptor that is an unsigned integer and can be used to trigger `EBADF` as the first argument. If no such file descriptor could be generated, a skip message will be printed and the `func` will not be run.

`skip(msg)`

- `msg` <string>

Logs '1..0 # Skipped:' + `msg` and exits with exit code `0`.

`skipIfDumbTerminal()`

Skip the rest of the tests if the current terminal is a dumb terminal

`skipIfEslintMissing()`

Skip the rest of the tests in the current file when `ESLint` is not available at `tools/node_modules/eslint`

`skipIfInspectorDisabled()`

Skip the rest of the tests in the current file when the Inspector was disabled at compile time.

`skipIf32Bits()`

Skip the rest of the tests in the current file when the Node.js executable was compiled with a pointer size smaller than 64 bits.

`skipIfWorker()`

Skip the rest of the tests in the current file when not running on a main thread.

ArrayStream Module

The `ArrayStream` module provides a simple `Stream` that pushes elements from a given array.

```js
const ArrayStream = require('../common/arraystream');
const stream = new ArrayStream();
stream.run(['a', 'b', 'c']);
```

It can be used within tests as a simple mock stream.

Countdown Module

The `Countdown` module provides a simple countdown mechanism for tests that require a particular action to be taken after a given number of completed tasks (for instance, shutting down an HTTP server after a specific number of requests). The Countdown will fail the test if the remainder did not reach 0.

```js
const Countdown = require('../common/countdown');

function doSomething() {
  console.log('.');
}

const countdown = new Countdown(2, doSomething);
countdown.dec();
countdown.dec();
```

`new Countdown(limit, callback)`

- `limit` {number}
- `callback` {function}

Creates a new `Countdown` instance.

`Countdown.prototype.dec()`

Decrements the `Countdown` counter.

`Countdown.prototype.remaining`

Specifies the remaining number of times `Countdown.prototype.dec()` must be called before the callback is invoked.

CPU Profiler module

The `cpu-prof` module provides utilities related to CPU profiling tests.

`env`

- Default: { ...process.env, NODE_DEBUG_NATIVE: 'INSPECTOR_PROFILER' }

Environment variables used in profiled processes.

`getCpuProfiles(dir)`

- `dir` {string} The directory containing the CPU profile files.
- return <string>

Returns an array of all `.cpuprofile` files found in `dir`.

`getFrames(file, suffix)`

- `file` {string} Path to a `.cpuprofile` file.
- `suffix` {string} Suffix of the URL of call frames to retrieve.
- returns { frames: <Object>, nodes: <Object> }

Returns an object containing an array of the relevant call frames and an array of all the profile nodes.

`kCpuProfInterval`

Sampling interval in microseconds.

`verifyFrames(output, file, suffix)`

- `output` {string}
- `file` {string}
- `suffix` {string}

Throws an `AssertionError` if there are no call frames with the expected `suffix` in the profiling data contained in `file`.

Debugger module

Provides common functionality for tests for `node inspect`.

`startCLI(args[[, flags], spawnOpts])`

- `args` <string>
- `flags` <string> default = []
- `showOpts` <Object> default = {}
- return <Object>

Returns a null-prototype object with properties that are functions and getters used to interact with the `node inspect` CLI. These functions are:

- `flushOutput()`
- `waitFor()`
- `waitForPrompt()`
- `waitForInitialBreak()`
- `breakInfo`
- `ctrlC()`
- `output`

- `rawOutput`
- `parseSourceLines()`
- `writeLine()`
- `command()`
- `stepCommand()`
- `quit()`

`DNS` Module

The `DNS` module provides utilities related to the `dns` built-in module.

`errorLookupMock(code, syscall)`

- `code` <string> Defaults to `dns.mockedErrorCode`.
- `syscall` <string> Defaults to `dns.mockedSysCall`.
- return <Function>

A mock for the `lookup` option of `net.connect()` that would result in an error with the `code` and the `syscall` specified. Returns a function that has the same signature as `dns.lookup()`.

`mockedErrorCode`

The default `code` of errors generated by `errorLookupMock`.

`mockedSysCall`

The default `syscall` of errors generated by `errorLookupMock`.

`readDomainFromPacket(buffer, offset)`

- `buffer` <Buffer>
- `offset` <number>
- return <Object>

Reads the domain string from a packet and returns an object containing the number of bytes read and the domain.

`parseDNSPacket(buffer)`

- `buffer` <Buffer>
- return <Object>

Parses a DNS packet. Returns an object with the values of the various flags of the packet depending on the type of packet.

`writeIPv6(ip)`

- `ip` <string>
- return <Buffer>

Reads an IPv6 String and returns a Buffer containing the parts.

`writeDomainName(domain)`

- `domain` <string>
- return <Buffer>

Reads a Domain String and returns a Buffer containing the domain.

`writeDNSPacket(parsed)`

- `parsed` <Object>
- return <Buffer>

Takes in a parsed Object and writes its fields to a DNS packet as a Buffer object.

Duplex pair helper

The `common/duplexpair` module exports a single function `makeDuplexPair`, which returns an object `{ clientSide, serverSide }` where each side is a `Duplex` stream connected to the other side.

There is no difference between client or server side beyond their names.

Environment variables

The behavior of the Node.js test suite can be altered using the following environment variables.

`NODE_COMMON_PORT`

If set, `NODE_COMMON_PORT`'s value overrides the `common.PORT` default value of 12346.

`NODE_SKIP_FLAG_CHECK`

If set, command line arguments passed to individual tests are not validated.

`NODE_SKIP_CRYPTO`

If set, crypto tests are skipped.

`NODE_TEST_KNOWN_GLOBALS`

A comma-separated list of variables names that are appended to the global variable allowlist. Alternatively, if `NODE_TEST_KNOWN_GLOBALS` is set to `'0'`, global leak detection is disabled.

Fixtures Module

The `common/fixtures` module provides convenience methods for working with files in the `test/fixtures` directory.

`fixtures.fixturesDir`

- <string>

The absolute path to the `test/fixtures/` directory.

`fixtures.path(...args)`

- `...args` <string>

Returns the result of `path.join(fixtures.fixturesDir, ...args)`.

`fixtures.readSync(args[, enc])`

- `args` <string> | <Array>

Returns the result of `fs.readFileSync(path.join(fixtures.fixturesDir, ...args), 'enc')`.

`fixtures.readKey(arg[, enc])`

- `arg` <string>

Returns the result of `fs.readFileSync(path.join(fixtures.fixturesDir, 'keys', arg), 'enc')`.

Heap dump checker module

This provides utilities for checking the validity of heap dumps. This requires the usage of `--expose-internals`.

`heap.recordState()`

Create a heap dump and an embedder graph copy for inspection. The returned object has a `validateSnapshotNodes` function similar to the one listed below. (`heap.validateSnapshotNodes(...)` is a shortcut for `heap.recordState().validateSnapshotNodes(...)`.)

`heap.validateSnapshotNodes(name, expected, options)`

- `name` <string> Look for this string as the name of heap dump nodes.
- `expected` <Array> A list of objects, possibly with an `children` property that points to expected other adjacent nodes.
- `options` <Array>
  - `loose` <boolean> Do not expect an exact listing of occurrences of nodes with name `name` in `expected`.

Create a heap dump and an embedder graph copy and validate occurrences.

```
validateSnapshotNodes('TLSWRAP', [
  {
    children: [
      { name: 'enc_out' },
      { name: 'enc_in' },
      { name: 'TLSWrap' },
    ]
  },
]);
```

hijackstdio Module

The `hijackstdio` module provides utility functions for temporarily redirecting `stdout` and `stderr` output.

```
const { hijackStdout, restoreStdout } = require('../common/hijackstdio');

hijackStdout((data) => {
  /* Do something with data */
  restoreStdout();
});

console.log('this is sent to the hijacked listener');
```

`hijackStderr(listener)`

- `listener` <Function>: a listener with a single parameter called `data`.

Eavesdrop to `process.stderr.write()` calls. Once `process.stderr.write()` is called, `listener` will also be called and the `data` of `write` function will be passed to `listener`. What's more, `process.stderr.writeTimes` is a count of the number of calls.

`hijackStdout(listener)`

- `listener` <Function>: a listener with a single parameter called `data`.

Eavesdrop to `process.stdout.write()` calls. Once `process.stdout.write()` is called, `listener` will also be called and the `data` of `write` function will be passed to `listener`. What's more, `process.stdout.writeTimes` is a count of the number of calls.

restoreStderr()

Restore the original `process.stderr.write()`. Used to restore `stderr` to its original state after calling `hijackstdio.hijackStdErr()`.

restoreStdout()

Restore the original `process.stdout.write()`. Used to restore `stdout` to its original state after calling `hijackstdio.hijackStdOut()`.

HTTP/2 Module

The http2.js module provides a handful of utilities for creating mock HTTP/2 frames for testing of HTTP/2 endpoints

```
const http2 = require('../common/http2');
```

Class: Frame

The `http2.Frame` is a base class that creates a `Buffer` containing a serialized HTTP/2 frame header.

```
// length is a 24-bit unsigned integer
// type is an 8-bit unsigned integer identifying the frame type
// flags is an 8-bit unsigned integer containing the flag bits
// id is the 32-bit stream identifier, if any.
const frame = new http2.Frame(length, type, flags, id);

// Write the frame data to a socket
socket.write(frame.data);
```

The serialized `Buffer` may be retrieved using the `frame.data` property.

Class: DataFrame extends Frame

The `http2.DataFrame` is a subclass of `http2.Frame` that serializes a `DATA` frame.

```
// id is the 32-bit stream identifier
// payload is a Buffer containing the DATA payload
// padlen is an 8-bit integer giving the number of padding bytes to include
// final is a boolean indicating whether the End-of-stream flag should be set,
// defaults to false.
const frame = new http2.DataFrame(id, payload, padlen, final);
```

```
socket.write(frame.data);
```

Class: HeadersFrame

The `http2.HeadersFrame` is a subclass of `http2.Frame` that serializes a `HEADERS` frame.

```
// id is the 32-bit stream identifier
// payload is a Buffer containing the HEADERS payload (see either
// http2.kFakeRequestHeaders or http2.kFakeResponseHeaders).
// padlen is an 8-bit integer giving the number of padding bytes to include
// final is a boolean indicating whether the End-of-stream flag should be set,
// defaults to false.
const frame = new http2.HeadersFrame(id, payload, padlen, final);

socket.write(frame.data);
```

Class: SettingsFrame

The `http2.SettingsFrame` is a subclass of `http2.Frame` that serializes an empty `SETTINGS` frame.

```
// ack is a boolean indicating whether or not to set the ACK flag.
const frame = new http2.SettingsFrame(ack);

socket.write(frame.data);
```

`http2.kFakeRequestHeaders`

Set to a `Buffer` instance that contains a minimal set of serialized HTTP/2 request headers to be used as the payload of a `http2.HeadersFrame`.

```
const frame = new http2.HeadersFrame(1, http2.kFakeRequestHeaders, 0, true);

socket.write(frame.data);
```

`http2.kFakeResponseHeaders`

Set to a `Buffer` instance that contains a minimal set of serialized HTTP/2 response headers to be used as the payload a `http2.HeadersFrame`.

```
const frame = new http2.HeadersFrame(1, http2.kFakeResponseHeaders, 0, true);

socket.write(frame.data);
```

`http2.kClientMagic`

Set to a `Buffer` containing the preamble bytes an HTTP/2 client must send upon initial establishment of a connection.

```
socket.write(http2.kClientMagic);
```

Internet Module

The `common/internet` module provides utilities for working with internet-related tests.

`internet.addresses`

- <Object>
  - `INET_HOST` <string> A generic host that has registered common DNS records, supports both IPv4 and IPv6, and provides basic HTTP/HTTPS services
  - `INET4_HOST` <string> A host that provides IPv4 services
  - `INET6_HOST` <string> A host that provides IPv6 services
  - `INET4_IP` <string> An accessible IPv4 IP, defaults to the Google Public DNS IPv4 address
  - `INET6_IP` <string> An accessible IPv6 IP, defaults to the Google Public DNS IPv6 address
  - `INVALID_HOST` <string> An invalid host that cannot be resolved
  - `MX_HOST` <string> A host with MX records registered
  - `SRV_HOST` <string> A host with SRV records registered
  - `PTR_HOST` <string> A host with PTR records registered
  - `NAPTR_HOST` <string> A host with NAPTR records registered
  - `SOA_HOST` <string> A host with SOA records registered
  - `CNAME_HOST` <string> A host with CNAME records registered
  - `NS_HOST` <string> A host with NS records registered
  - `TXT_HOST` <string> A host with TXT records registered
  - `DNS4_SERVER` <string> An accessible IPv4 DNS server
  - `DNS6_SERVER` <string> An accessible IPv6 DNS server

A set of addresses for internet-related tests. All properties are configurable via `NODE_TEST_*` environment variables. For example, to configure `internet.addresses.INET_HOST`, set the environment variable `NODE_TEST_INET_HOST` to a specified host.

ongc Module

The `ongc` module allows a garbage collection listener to be installed. The module exports a single `onGC()` function.

```
require('../common');
const onGC = require('../common/ongc');
```

```
onGC({}, { ongc() { console.log('collected'); } });
```

`onGC(target, listener)`

- `target` <Object>
- `listener` <Object>
  - `ongc` <Function>

Installs a GC listener for the collection of `target`.

This uses `async_hooks` for GC tracking. This means that it enables `async_hooks` tracking, which may affect the test functionality. It also means that between a `global.gc()` call and the listener being invoked a full `setImmediate()` invocation passes.

`listener` is an object to make it easier to use a closure; the target object should not be in scope when `listener.ongc()` is created.

Report Module

The `report` module provides helper functions for testing diagnostic reporting functionality.

`findReports(pid, dir)`

- `pid` <number> Process ID to retrieve diagnostic report files for.
- `dir` <string> Directory to search for diagnostic report files.
- return <Array>

Returns an array of diagnostic report file names found in `dir`. The files should have been generated by a process whose PID matches `pid`.

`validate(filepath)`

- `filepath` <string> Diagnostic report filepath to validate.

Validates the schema of a diagnostic report file whose path is specified in `filepath`. If the report fails validation, an exception is thrown.

`validateContent(report)`

- `report` <Object> | <string> JSON contents of a diagnostic report file, the parsed Object thereof, or the result of `process.report.getReport()`.

Validates the schema of a diagnostic report whose content is specified in `report`. If the report fails validation, an exception is thrown.

tick Module

The `tick` module provides a helper function that can be used to call a callback after a given number of event loop "ticks".

`tick(x, cb)`

- `x` <number> Number of event loop "ticks".
- `cb` <Function> A callback function.

tmpdir Module

The `tmpdir` module supports the use of a temporary directory for testing.

`path`

- <string>

The realpath of the testing temporary directory.

`refresh()`

Deletes and recreates the testing temporary directory.

The first time `refresh()` runs, it adds a listener to process `'exit'` that cleans the temporary directory. Thus, every file under `tmpdir.path` needs to be closed before the test completes. A good way to do this is to add a listener to process `'beforeExit'`. If a file needs to be left open until Node.js completes, use a child process and call `refresh()` only in the parent.

It is usually only necessary to call `refresh()` once in a test file. Avoid calling it more than once in an asynchronous context as one call might refresh the temporary directory of a different context, causing the test to fail somewhat mysteriously.

UDP pair helper

The `common/udppair` module exports a function `makeUDPPair` and a class `FakeUDPWrap`.

`FakeUDPWrap` emits `'send'` events when data is to be sent on it, and provides an `emitReceived()` API for actin as if data has been received on it.

`makeUDPPair` returns an object `{ clientSide, serverSide }` where each side is an `FakeUDPWrap` connected to the other side.

There is no difference between client or server side beyond their names.

WPT Module

`harness`

A legacy port of Web Platform Tests harness.

See the source code for definitions. Please avoid using it in new code - the current usage of this port in tests is being migrated to the original WPT harness, see the WPT tests README.

Class: WPTRunner

A driver class for running WPT with the WPT harness in a worker thread.

See the WPT tests README for details.