Diagnostic report

Stability: 2 - Stable

Delivers a JSON-formatted diagnostic summary, written to a file.

The report is intended for development, test and production use, to capture and preserve information for problem determination. It includes JavaScript and native stack traces, heap statistics, platform information, resource usage etc. With the report option enabled, diagnostic reports can be triggered on unhandled exceptions, fatal errors and user signals, in addition to triggering programmatically through API calls.

A complete example report that was generated on an uncaught exception is provided below for reference.


```
{
  "header": {
    "reportVersion": 1,
    "event": "exception",
    "trigger": "Exception",
    "filename": "report.20181221.005011.8974.0.001.json",
    "dumpEventTime": "2018-12-21T00:50:11Z",
    "dumpEventTimeStamp": "1545371411331",
    "processId": 8974,
    "cwd": "/home/nodeuser/project/node",
    "commandLine": [
      "/home/nodeuser/project/node/out/Release/node",
      "--report-uncaught-exception",
      "/home/nodeuser/project/node/test/report/test-exception.js",
      "child"
    ],
    "nodejsVersion": "v12.0.0-pre",
    "glibcVersionRuntime": "2.17",
    "glibcVersionCompiler": "2.17",
    "wordSize": "64 bit",
    "arch": "x64",
    "platform": "linux",
    "componentVersions": {
      "node": "12.0.0-pre",
      "v8": "7.1.302.28-node.5",
      "uv": "1.24.1",
      "zlib": "1.2.11",
      "ares": "1.15.0",
      "modules": "68",
      "nghttp2": "1.34.0",
      "napi": "3",
      "llhttp": "1.0.1",
```

```
    "openssl": "1.1.0j"
  },
  "release": {
    "name": "node"
  },
  "osName": "Linux",
  "osRelease": "3.10.0-862.el7.x86_64",
  "osVersion": "#1 SMP Wed Mar 21 18:14:51 EDT 2018",
  "osMachine": "x86_64",
  "cpus": [
    {
      "model": "Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz",
      "speed": 2700,
      "user": 88902660,
      "nice": 0,
      "sys": 50902570,
      "idle": 241732220,
      "irq": 0
    },
    {
      "model": "Intel(R) Core(TM) i7-6820HQ CPU @ 2.70GHz",
      "speed": 2700,
      "user": 88902660,
      "nice": 0,
      "sys": 50902570,
      "idle": 241732220,
      "irq": 0
    }
  ],
  "networkInterfaces": [
    {
      "name": "en0",
      "internal": false,
      "mac": "13:10:de:ad:be:ef",
      "address": "10.0.0.37",
      "netmask": "255.255.255.0",
      "family": "IPv4"
    }
  ],
  "host": "test_machine"
},
"javascriptStack": {
  "message": "Error: *** test-exception.js: throwing uncaught Error",
  "stack": [
    "at myException (/home/nodeuser/project/node/test/report/test-exception.js:9:11)",
    "at Object.<anonymous> (/home/nodeuser/project/node/test/report/test-exception.js:12:3
```

```
    "at Module._compile (internal/modules/cjs/loader.js:718:30)",
    "at Object.Module._extensions..js (internal/modules/cjs/loader.js:729:10)",
    "at Module.load (internal/modules/cjs/loader.js:617:32)",
    "at tryModuleLoad (internal/modules/cjs/loader.js:560:12)",
    "at Function.Module._load (internal/modules/cjs/loader.js:552:3)",
    "at Function.Module.runMain (internal/modules/cjs/loader.js:771:12)",
    "at executeUserCode (internal/bootstrap/node.js:332:15)"
  ]
},
"nativeStack": [
  {
    "pc": "0x000055b57f07a9ef",
    "symbol": "report::GetNodeReport(v8::Isolate*, node::Environment*, char const*, char c
  },
  {
    "pc": "0x000055b57f07cf03",
    "symbol": "report::GetReport(v8::FunctionCallbackInfo<v8::Value> const&) [./node]"
  },
  {
    "pc": "0x000055b57f1bccfd",
    "symbol": " [./node]"
  },
  {
    "pc": "0x000055b57f1be048",
    "symbol": "v8::internal::Builtin_HandleApiCall(int, v8::internal::Object**, v8::interr
  },
  {
    "pc": "0x000055b57feeda0e",
    "symbol": " [./node]"
  }
],
"javascriptHeap": {
  "totalMemory": 6127616,
  "totalCommittedMemory": 4357352,
  "usedMemory": 3221136,
  "availableMemory": 1521370240,
  "memoryLimit": 1526909922,
  "heapSpaces": {
    "read_only_space": {
      "memorySize": 524288,
      "committedMemory": 39208,
      "capacity": 515584,
      "used": 30504,
      "available": 485080
    },
    "new_space": {
```

```
      "memorySize": 2097152,
      "committedMemory": 2019312,
      "capacity": 1031168,
      "used": 985496,
      "available": 45672
    },
    "old_space": {
      "memorySize": 2273280,
      "committedMemory": 1769008,
      "capacity": 1974640,
      "used": 1725488,
      "available": 249152
    },
    "code_space": {
      "memorySize": 696320,
      "committedMemory": 184896,
      "capacity": 152128,
      "used": 152128,
      "available": 0
    },
    "map_space": {
      "memorySize": 536576,
      "committedMemory": 344928,
      "capacity": 327520,
      "used": 327520,
      "available": 0
    },
    "large_object_space": {
      "memorySize": 0,
      "committedMemory": 0,
      "capacity": 1520590336,
      "used": 0,
      "available": 1520590336
    },
    "new_large_object_space": {
      "memorySize": 0,
      "committedMemory": 0,
      "capacity": 0,
      "used": 0,
      "available": 0
    }
  }
},
"resourceUsage": {
  "userCpuSeconds": 0.069595,
  "kernelCpuSeconds": 0.019163,
```

```
    "cpuConsumptionPercent": 0.000000,
    "maxRss": 18079744,
    "pageFaults": {
      "IORequired": 0,
      "IONotRequired": 4610
    },
    "fsActivity": {
      "reads": 0,
      "writes": 0
    }
  },
  "uvthreadResourceUsage": {
    "userCpuSeconds": 0.068457,
    "kernelCpuSeconds": 0.019127,
    "cpuConsumptionPercent": 0.000000,
    "fsActivity": {
      "reads": 0,
      "writes": 0
    }
  },
  "libuv": [
    {
      "type": "async",
      "is_active": true,
      "is_referenced": false,
      "address": "0x0000000102910900",
      "details": ""
    },
    {
      "type": "timer",
      "is_active": false,
      "is_referenced": false,
      "address": "0x00007fff5fbfeab0",
      "repeat": 0,
      "firesInMsFromNow": 94403548320796,
      "expired": true
    },
    {
      "type": "check",
      "is_active": true,
      "is_referenced": false,
      "address": "0x00007fff5fbfeb48"
    },
    {
      "type": "idle",
      "is_active": false,
```

```
  "is_referenced": true,
  "address": "0x00007fff5fbfebc0"
},
{
  "type": "prepare",
  "is_active": false,
  "is_referenced": false,
  "address": "0x00007fff5fbfec38"
},
{
  "type": "check",
  "is_active": false,
  "is_referenced": false,
  "address": "0x00007fff5fbfecb0"
},
{
  "type": "async",
  "is_active": true,
  "is_referenced": false,
  "address": "0x000000010188f2e0"
},
{
  "type": "tty",
  "is_active": false,
  "is_referenced": true,
  "address": "0x000055b581db0e18",
  "width": 204,
  "height": 55,
  "fd": 17,
  "writeQueueSize": 0,
  "readable": true,
  "writable": true
},
{
  "type": "signal",
  "is_active": true,
  "is_referenced": false,
  "address": "0x000055b581d80010",
  "signum": 28,
  "signal": "SIGWINCH"
},
{
  "type": "tty",
  "is_active": true,
  "is_referenced": true,
  "address": "0x000055b581df59f8",
```

```
      "width": 204,
      "height": 55,
      "fd": 19,
      "writeQueueSize": 0,
      "readable": true,
      "writable": true
    },
    {
      "type": "loop",
      "is_active": true,
      "address": "0x000055fc7b2cb180",
      "loopIdleTimeSeconds": 22644.8
    }
  ],
  "workers": [],
  "environmentVariables": {
    "REMOTEHOST": "REMOVED",
    "MANPATH": "/opt/rh/devtoolset-3/root/usr/share/man:",
    "XDG_SESSION_ID": "66126",
    "HOSTNAME": "test_machine",
    "HOST": "test_machine",
    "TERM": "xterm-256color",
    "SHELL": "/bin/csh",
    "SSH_CLIENT": "REMOVED",
    "PERL5LIB": "/opt/rh/devtoolset-3/root//usr/lib64/perl5/vendor_perl:/opt/rh/devtoolset-3
    "OLDPWD": "/home/nodeuser/project/node/src",
    "JAVACONFDIRS": "/opt/rh/devtoolset-3/root/etc/java:/etc/java",
    "SSH_TTY": "/dev/pts/0",
    "PCP_DIR": "/opt/rh/devtoolset-3/root",
    "GROUP": "normaluser",
    "USER": "nodeuser",
    "LD_LIBRARY_PATH": "/opt/rh/devtoolset-3/root/usr/lib64:/opt/rh/devtoolset-3/root/usr/li
    "HOSTTYPE": "x86_64-linux",
    "XDG_CONFIG_DIRS": "/opt/rh/devtoolset-3/root/etc/xdg:/etc/xdg",
    "MAIL": "/var/spool/mail/nodeuser",
    "PATH": "/home/nodeuser/project/node:/opt/rh/devtoolset-3/root/usr/bin:/usr/local/bin:/u
    "PWD": "/home/nodeuser/project/node",
    "LANG": "en_US.UTF-8",
    "PS1": "\\u@\\h : \\[\\e[31m\\]\\w\\[\\e[m\\] >  ",
    "SHLVL": "2",
    "HOME": "/home/nodeuser",
    "OSTYPE": "linux",
    "VENDOR": "unknown",
    "PYTHONPATH": "/opt/rh/devtoolset-3/root/usr/lib64/python2.7/site-packages:/opt/rh/devto
    "MACHTYPE": "x86_64",
    "LOGNAME": "nodeuser",
```

```
    "XDG_DATA_DIRS": "/opt/rh/devtoolset-3/root/usr/share:/usr/local/share:/usr/share",
    "LESSOPEN": "||/usr/bin/lesspipe.sh %s",
    "INFOPATH": "/opt/rh/devtoolset-3/root/usr/share/info",
    "XDG_RUNTIME_DIR": "/run/user/50141",
    "_": "./node"
  },
  "userLimits": {
    "core_file_size_blocks": {
      "soft": "",
      "hard": "unlimited"
    },
    "data_seg_size_kbytes": {
      "soft": "unlimited",
      "hard": "unlimited"
    },
    "file_size_blocks": {
      "soft": "unlimited",
      "hard": "unlimited"
    },
    "max_locked_memory_bytes": {
      "soft": "unlimited",
      "hard": 65536
    },
    "max_memory_size_kbytes": {
      "soft": "unlimited",
      "hard": "unlimited"
    },
    "open_files": {
      "soft": "unlimited",
      "hard": 4096
    },
    "stack_size_bytes": {
      "soft": "unlimited",
      "hard": "unlimited"
    },
    "cpu_time_seconds": {
      "soft": "unlimited",
      "hard": "unlimited"
    },
    "max_user_processes": {
      "soft": "unlimited",
      "hard": 4127290
    },
    "virtual_memory_kbytes": {
      "soft": "unlimited",
      "hard": "unlimited"
```

```
    }
  },
  "sharedObjects": [
    "/lib64/libdl.so.2",
    "/lib64/librt.so.1",
    "/lib64/libstdc++.so.6",
    "/lib64/libm.so.6",
    "/lib64/libgcc_s.so.1",
    "/lib64/libpthread.so.0",
    "/lib64/libc.so.6",
    "/lib64/ld-linux-x86-64.so.2"
  ]
}
```

Usage

```
node --report-uncaught-exception --report-on-signal \
--report-on-fatalerror app.js
```

- `--report-uncaught-exception` Enables report to be generated on uncaught exceptions. Useful when inspecting JavaScript stack in conjunction with native stack and other runtime environment data.
- `--report-on-signal` Enables report to be generated upon receiving the specified (or predefined) signal to the running Node.js process. (See below on how to modify the signal that triggers the report.) Default signal is `SIGUSR2`. Useful when a report needs to be triggered from another program. Application monitors may leverage this feature to collect report at regular intervals and plot rich set of internal runtime data to their views.

Signal based report generation is not supported in Windows.

Under normal circumstances, there is no need to modify the report triggering signal. However, if `SIGUSR2` is already used for other purposes, then this flag helps to change the signal for report generation and preserve the original meaning of `SIGUSR2` for the said purposes.

- `--report-on-fatalerror` Enables the report to be triggered on fatal errors (internal errors within the Node.js runtime, such as out of memory) that leads to termination of the application. Useful to inspect various diagnostic data elements such as heap, stack, event loop state, resource consumption etc. to reason about the fatal error.
- `--report-compact` Write reports in a compact format, single-line JSON, more easily consumable by log processing systems than the default multi-line format designed for human consumption.
- `--report-directory` Location at which the report will be generated.
- `--report-filename` Name of the file to which the report will be written.

- `--report-signal` Sets or resets the signal for report generation (not supported on Windows). Default signal is `SIGUSR2`.

A report can also be triggered via an API call from a JavaScript application:

```
process.report.writeReport();
```

This function takes an optional additional argument `filename`, which is the name of a file into which the report is written.

```
process.report.writeReport('./foo.json');
```

This function takes an optional additional argument `err` which is an `Error` object that will be used as the context for the JavaScript stack printed in the report. When using report to handle errors in a callback or an exception handler, this allows the report to include the location of the original error as well as where it was handled.

```
try {
  process.chdir('/non-existent-path');
} catch (err) {
  process.report.writeReport(err);
}
// Any other code
```

If both filename and error object are passed to `writeReport()` the error object must be the second parameter.

```
try {
  process.chdir('/non-existent-path');
} catch (err) {
  process.report.writeReport(filename, err);
}
// Any other code
```

The content of the diagnostic report can be returned as a JavaScript Object via an API call from a JavaScript application:

```
const report = process.report.getReport();
console.log(typeof report === 'object'); // true

// Similar to process.report.writeReport() output
console.log(JSON.stringify(report, null, 2));
```

This function takes an optional additional argument `err`, which is an `Error` object that will be used as the context for the JavaScript stack printed in the report.

```
const report = process.report.getReport(new Error('custom error'));
console.log(typeof report === 'object'); // true
```

The API versions are useful when inspecting the runtime state from within the application, in expectation of self-adjusting the resource consumption, load balancing, monitoring etc.

The content of the report consists of a header section containing the event type, date, time, PID and Node.js version, sections containing JavaScript and native stack traces, a section containing V8 heap information, a section containing `libuv` handle information and an OS platform information section showing CPU and memory usage and system limits. An example report can be triggered using the Node.js REPL:

```
$ node
> process.report.writeReport();
Writing Node.js report to file: report.20181126.091102.8480.0.001.json
Node.js report completed
>
```

When a report is written, start and end messages are issued to stderr and the filename of the report is returned to the caller. The default filename includes the date, time, PID and a sequence number. The sequence number helps in associating the report dump with the runtime state if generated multiple times for the same Node.js process.

Configuration

Additional runtime configuration of report generation is available via the following properties of `process.report`:

`reportOnFatalError` triggers diagnostic reporting on fatal errors when `true`. Defaults to `false`.

`reportOnSignal` triggers diagnostic reporting on signal when `true`. This is not supported on Windows. Defaults to `false`.

`reportOnUncaughtException` triggers diagnostic reporting on uncaught exception when `true`. Defaults to `false`.

`signal` specifies the POSIX signal identifier that will be used to intercept external triggers for report generation. Defaults to `'SIGUSR2'`.

`filename` specifies the name of the output file in the file system. Special meaning is attached to `stdout` and `stderr`. Usage of these will result in report being written to the associated standard streams. In cases where standard streams are used, the value in `directory` is ignored. URLs are not supported. Defaults to a composite filename that contains timestamp, PID and sequence number.

`directory` specifies the filesystem directory where the report will be written. URLs are not supported. Defaults to the current working directory of the Node.js process.

```
// Trigger report only on uncaught exceptions.
process.report.reportOnFatalError = false;
process.report.reportOnSignal = false;
process.report.reportOnUncaughtException = true;

// Trigger report for both internal errors as well as external signal.
process.report.reportOnFatalError = true;
process.report.reportOnSignal = true;
process.report.reportOnUncaughtException = false;

// Change the default signal to 'SIGQUIT' and enable it.
process.report.reportOnFatalError = false;
process.report.reportOnUncaughtException = false;
process.report.reportOnSignal = true;
process.report.signal = 'SIGQUIT';
```

Configuration on module initialization is also available via environment variables:

```
NODE_OPTIONS="--report-uncaught-exception \
  --report-on-fatalerror --report-on-signal \
  --report-signal=SIGUSR2  --report-filename=./report.json \
  --report-directory=/home/nodeuser"
```

Specific API documentation can be found under `process API documentation` section.

Interaction with workers

`Worker` threads can create reports in the same way that the main thread does.

Reports will include information on any Workers that are children of the current thread as part of the `workers` section, with each Worker generating a report in the standard report format.

The thread which is generating the report will wait for the reports from Worker threads to finish. However, the latency for this will usually be low, as both running JavaScript and the event loop are interrupted to generate the report.