readme.md for @push.rocks/smartsystem

Smart System Interaction for Node.js — your unified gateway to hardware, OS, environment, and network intelligence.

> Zero-hassle system information and environment management for modern Node.js applications.

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

□□ Features

`@push.rocks/smartsystem` consolidates system interaction into a single, elegant TypeScript-first API:

- □□ **Unified System Interface** — One `Smartsystem` instance to rule them all
- □□ **Runtime Detection** — Identify Node.js, Deno, Bun, or browser environments via @push.rocks/smartenv
- □□ **CPU Information** — Direct access to CPU core specs without the boilerplate
- □□ **Network Intelligence** — Port scanning, speed tests, DNS resolution, traceroutes, and more via @push.rocks/smartnetwork
- □□ **Deep System Insights** — Comprehensive hardware, OS, Docker, and performance data via systeminformation
- ⚡ **TypeScript Native** — Built with TypeScript, for TypeScript (but works great with JS too!)

Installation

```
# Using pnpm (recommended)
pnpm add @push.rocks/smartsystem

# Using npm
npm install @push.rocks/smartsystem
```

Quick Start

```
import { Smartsystem } from '@push.rocks/smartsystem';

// Create your system interface
const mySystem = new Smartsystem();

// CPU cores at a glance
console.log(`Running on ${mySystem.cpus.length} CPU cores`);

// Deep system data
const osInfo = await mySystem.information.osInfo();
console.log(`OS: ${osInfo.distro} ${osInfo.release}`);
```

API Overview

The `Smartsystem` class provides these powerful properties:

Property	Type	Description
`env`	`Smartenv`	Runtime environment detection — identify Node.js, Deno, Bun, browser, CI, and OS platform
`cpus`	`os.CpuInfo[]`	Direct access to CPU core information (model, speed, times)
`network`	`SmartNetwork`	Network utilities — port scanning, speed tests, DNS, ping, traceroute

Property	Type	Description
`information`	`systeminformation`	Full access to 50+ system data functions (CPU, memory, disk, GPU, Docker, etc.)

□□ Usage Examples

□□ Runtime Environment Detection

Detect where your code is running and adapt accordingly:

```
const mySystem = new Smartsystem();

// Runtime identification
console.log(`Runtime: ${mySystem.env.runtimeEnv}`); // 'node', 'deno', 'bun', or 'browser'
console.log(`Is Node.js: ${mySystem.env.isNode}`);
console.log(`Is CI: ${mySystem.env.isCI}`);

// OS detection
if (await mySystem.env.isLinuxAsync()) {
  console.log('□□Running on Linux');
} else if (await mySystem.env.isMacAsync()) {
  console.log('□□Running on macOS');
}

// Conditionally load modules based on runtime
const fsModule = await mySystem.env.getSafeModuleFor<typeof import('fs')>('node', 'fs');
```

□□ CPU Information at Your Fingertips

```
const mySystem = new Smartsystem();

console.log(`CPU Cores: ${mySystem.cpus.length}`);
console.log(`Primary CPU: ${mySystem.cpus[0].model}`);
console.log(`Speed: ${mySystem.cpus[0].speed} MHz`);
```

Network Intelligence

Leverage the full power of `SmartNetwork`:

```
const mySystem = new Smartsystem();

// Check if a local port is available
const portFree = await mySystem.network.isLocalPortUnused(3000);
console.log(`Port 3000 available: ${portFree}`);

// Find a free port in a range
const freePort = await mySystem.network.findFreePort(8000, 9000);
console.log(`Free port found: ${freePort}`);

// Check remote port availability
const isReachable = await mySystem.network.isRemotePortAvailable('example.com', { port: 443
});
console.log(`Remote HTTPS reachable: ${isReachable}`);

// DNS resolution
const dns = await mySystem.network.resolveDns('example.com');
console.log(`A records: ${dns.A.join(', ')}`);

// Ping a host
const pingResult = await mySystem.network.ping('8.8.8.8', { count: 3 });

// Get public IP addresses
const publicIps = await mySystem.network.getPublicIps();
console.log(`Public IPv4: ${publicIps.v4}`);

// Run a speed test
const speed = await mySystem.network.getSpeed();
console.log(`Download: ${speed.downloadSpeed} | Upload: ${speed.uploadSpeed}`);

// HTTP endpoint health check
const health = await mySystem.network.checkEndpoint('https://example.com');
console.log(`Status: ${health.status}, RTT: ${health.rtt}ms`);

// Traceroute
```

```
const hops = await mySystem.network.traceroute('example.com', { maxHops: 15 });
hops.forEach(hop => console.log(`TTL ${hop.ttl}: ${hop.ip} (${hop.rtt}ms)`));
```

□□ Deep System Insights

Access detailed system information via `systeminformation`:

```
const mySystem = new Smartsystem();

// Hardware information
const systemInfo = await mySystem.information.system();
console.log(`System: ${systemInfo.manufacturer} ${systemInfo.model}`);

// Operating system details
const osInfo = await mySystem.information.osInfo();
console.log(`OS: ${osInfo.distro} ${osInfo.release}`);

// Real-time CPU metrics
const load = await mySystem.information.currentLoad();
console.log(`CPU Load: ${load.currentLoad.toFixed(2)}%`);

// Memory usage
const mem = await mySystem.information.mem();
console.log(`Memory: ${(mem.used / mem.total * 100).toFixed(2)}% used`);

// Disk information
const disk = await mySystem.information.fsSize();
disk.forEach(fs => console.log(`${fs.mount}: ${fs.use.toFixed(1)}% used`));

// GPU details
const gpu = await mySystem.information.graphics();
gpu.controllers.forEach(c => console.log(`GPU: ${c.model} (${c.vram}MB VRAM)`));

// Network interfaces
const defaultIface = await mySystem.information.networkInterfaceDefault();
console.log(`Default network interface: ${defaultIface}`);
```

□□ Real-World Use Cases

System Health Monitoring

Build a health check endpoint for your application:

```
const mySystem = new Smartsystem();

async function healthCheck() {
  const load = await mySystem.information.currentLoad();
  const mem = await mySystem.information.mem();
  const disk = await mySystem.information.fsSize();

  return {
    status: 'healthy',
    metrics: {
      cpuLoad: `${load.currentLoad.toFixed(2)}%`,
      memoryUsage: `${(mem.used / mem.total * 100).toFixed(2)}%`,
      diskUsage: disk.map(fs => ({
        mount: fs.mount,
        usage: `${fs.use.toFixed(2)}%`
      }))
    }
  };
}
```

Network Traffic Monitoring

Track bandwidth in real time:

```
const mySystem = new Smartsystem();

async function monitorNetwork() {
  const defaultInterface = await mySystem.information.networkInterfaceDefault();
  await mySystem.information.networkStats(defaultInterface); // baseline

  await new Promise(resolve => setTimeout(resolve, 1000));
```

```
  const current = await mySystem.information.networkStats(defaultInterface);
  const rxSpeed = (current[0].rx_sec / 1024 / 1024).toFixed(2);
  const txSpeed = (current[0].tx_sec / 1024 / 1024).toFixed(2);

  console.log(`□□Download: ${rxSpeed} MB/s | □□Upload: ${txSpeed} MB/s`);
}
```

Smart Resource Scaling

Dynamically scale workers based on available resources:

```
const mySystem = new Smartsystem();

async function getOptimalWorkerCount() {
  const cpuCount = mySystem.cpus.length;
  const load = await mySystem.information.currentLoad();
  const mem = await mySystem.information.mem();

  const memoryConstraint = Math.floor(mem.available / (512 * 1024 * 1024)); // 512MB per
worker
  const cpuConstraint = Math.max(1, Math.floor(cpuCount * (1 - load.currentLoad / 100)));

  return Math.min(memoryConstraint, cpuConstraint, cpuCount);
}
```

□□ Docker & Container Awareness

Inspect Docker state from within your app:

```
const mySystem = new Smartsystem();

const dockerInfo = await mySystem.information.dockerInfo();
console.log(`Docker containers: ${dockerInfo.containers} (${dockerInfo.containersRunning}
running)`);

const containers = await mySystem.information.dockerContainers();
containers.forEach(c => console.log(`  ${c.name}: ${c.state}`));
```

Cross-Platform Logic

Write platform-aware code cleanly:

```
const mySystem = new Smartsystem();

if (await mySystem.env.isLinuxAsync()) {
  // Linux-specific setup
} else if (await mySystem.env.isMacAsync()) {
  // macOS-specific setup
} else if (await mySystem.env.isWindowsAsync()) {
  // Windows-specific setup
}
```

□□ TypeScript Support

Full TypeScript support with comprehensive type definitions:

```
import { Smartsystem } from '@push.rocks/smartsystem';
import type { Systeminformation } from 'systeminformation';

const mySystem = new Smartsystem();

// All methods are fully typed
const cpuInfo: Systeminformation.CpuData = await mySystem.information.cpu();
const memInfo: Systeminformation.MemData = await mySystem.information.mem();
```

□□ API Documentation

For complete API documentation, visit: https://code.foss.global/push.rocks/smartsystem

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH
Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:12:38 UTC by foss.global Team
Updated 2026-03-28 12:19:25 UTC by foss.global Team