readme.md for @git.zone/tspm

TypeScript Process Manager — A robust, no-fuss process manager built for the modern TypeScript and Node.js ecosystem. Production-ready process management without the bloat.

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.

□□ What is TSPM?

TSPM is your production-ready process manager that handles the hard parts of running Node.js applications. Think PM2, but built from scratch for the TypeScript-first ecosystem with better memory management, intelligent logging, a clean daemon architecture, and native ESM support.

□ Key Features

- □□ **Smart Memory Management** — Tracks memory across entire process trees (parent + children), enforces limits, and auto-restarts on OOM
- □□ **Persistent Log Storage** — 10MB in-memory ring buffer per process, auto-persists to disk on stop/restart/crash
- □□ **Intelligent Auto-Restart** — Crashed processes restart with incremental backoff (1s → 10s), auto-stop after 10 consecutive failures
- □□ **File Watching** — Auto-restart on file changes for seamless development workflows
- □□ **Process Tree Tracking** — Monitors parent and all child processes as a unit — no orphans, ever
- □□ **Daemon Architecture** — Persistent background service that survives terminal sessions via Unix socket IPC

- □□ **Beautiful CLI** — Clean, informative output with table views, real-time log streaming, and search
- ⚡ **Zero Config** — Works out of the box; customize only when you need to
- □□ **Systemd Integration** — Run as a system service for production deployments with `tspm enable`
- □□ **Crash Log Reports** — Detailed crash reports with metadata, memory snapshots, and log history

□□ Installation

```
# Global install (recommended)
pnpm add -g @git.zone/tspm

# Or with npm
npm install -g @git.zone/tspm

# Or as a project dependency
pnpm add --save-dev @git.zone/tspm
```

□□ Quick Start

```
# Start the daemon
tspm daemon start

# Add a process
tspm add "node server.js" --name my-server --memory 1GB

# Start it
tspm start name:my-server

# See what's running
tspm list

# View logs
tspm logs name:my-server
```

```
# Stop it
tspm stop name:my-server
```

□□ CLI Reference

Targeting Processes

Most commands accept flexible process targeting:

Format	Example	Description
Numeric ID	`tspm start 1`	Direct ID reference
`id:N`	`tspm start id:1`	Explicit ID prefix
`name:LABEL`	`tspm start name:api`	Target by name

Use `tspm search <query>` to find processes by name or ID substring.

Process Management

`tspm add <command> [options]`

Register a new process configuration (without starting it).

Option	Description	Default
`--name <name>`	Process name	command string
`--memory <size>`	Memory limit (e.g. `512MB`, `2GB`)	`512MB`
`--cwd <path>`	Working directory	current directory
`--watch`	Enable file watching	`false`
`--watch-paths <paths>`	Comma-separated watch paths	—
`--autorestart`	Auto-restart on crash	`true`
`-i, --interactive`	Enter interactive edit after adding	—

```
# Simple Node.js app
tspm add "node server.js" --name api-server

# TypeScript with 2GB memory limit
```

```
tspm add "tsx src/index.ts" --name production-api --memory 2GB

# Dev mode with file watching
tspm add "tsx watch src/index.ts" --name dev-server --watch --watch-paths "src,config"

# One-shot worker (no auto-restart)
tspm add "node worker.js" --name batch-job --autorestart false

# Add + interactive edit
tspm add "node server.js" --name api -i
```

`tspm start <target>`

Start a registered process.

```
tspm start name:my-server
tspm start id:1
tspm start 1          # bare numeric id also works
```

`tspm stop <target>`

Gracefully stop a process (SIGTERM → 5s grace → SIGKILL).

```
tspm stop name:my-server
```

`tspm restart <target>`

Stop and restart a process, preserving its configuration.

```
tspm restart name:my-server
```

`tspm delete <target>`

Stop, remove from management, and delete persisted logs.

```
tspm delete name:old-server
```

`tspm edit <target>`

Interactively modify a process configuration (name, command, memory, etc.).

```
tspm edit name:my-server
```

`tspm search <query>`

Search processes by name or ID substring.

```
tspm search api
# Matches for "api":
#   id:3    name:api-server
```

Monitoring

`tspm list`

Display all managed processes in a table.

```
┌─────┬──────────────┬──────────┬───────┬────────────┬────────────┐
│ ID  │ Name         │ Status   │ PID   │ Memory     │ Restarts   │
├─────┼──────────────┼──────────┼───────┼────────────┼────────────┤
│ 1   │ my-app       │ online   │ 45123 │ 245.3 MB   │ 0          │
│ 2   │ worker       │ online   │ 45456 │ 128.7 MB   │ 2          │
│ 3   │ api-server   │ stopped  │ -     │ 0 B        │ 5          │
└─────┴──────────────┴──────────┴───────┴────────────┴────────────┘
```

`tspm describe <target>`

Detailed information about a specific process.

```
tspm describe name:my-server


# Process Details: my-server
# ──────────────────────────────────────────
# Status:       online
# PID:          45123
# Memory:       245.3 MB
# Uptime:       3600s
# Restarts:     0
#
# Configuration:
# ──────────────────────────────────────────
# Command:      node server.js
# Directory:    /home/user/project
```

```
# Memory Limit: 2 GB
# Auto-restart: true
# Watch:        disabled
```

`tspm logs <target> [options]`

View and stream process logs.

Option	Description	Default
`--lines <n>`	Number of lines	`50`
`--since <dur>`	Time filter (`10m`, `2h`, `1d`)	—
`--stderr-only`	Only stderr	—
`--stdout-only`	Only stdout	—
`--ndjson`	Output as newline-delimited JSON	—
`--follow`	Real-time streaming (like `tail -f`)	—

```
# View last 50 lines
tspm logs name:my-server

# Last 100 lines of stderr only
tspm logs name:my-server --lines 100 --stderr-only

# Stream logs in real time
tspm logs name:my-server --follow

# NDJSON output since 10 minutes ago
tspm logs name:my-server --since 10m --ndjson
```

Batch Operations

```
tspm start-all      # Start all saved processes
tspm stop-all       # Stop all running processes
tspm restart-all    # Restart all running processes
tspm reset          # ⚠ Stop all + clear all configs (prompts for confirmation)
```

Daemon Management

The daemon is a persistent background service that manages all processes. It starts automatically when needed.

```
tspm daemon start     # Start the daemon
tspm daemon stop      # Stop daemon + all managed processes
tspm daemon restart   # Restart daemon (preserves processes)
tspm daemon status    # Check daemon health + stats
```

System Service (systemd)

```
sudo tspm enable      # Install + enable as systemd service (auto-start on boot)
sudo tspm disable     # Remove systemd service
```

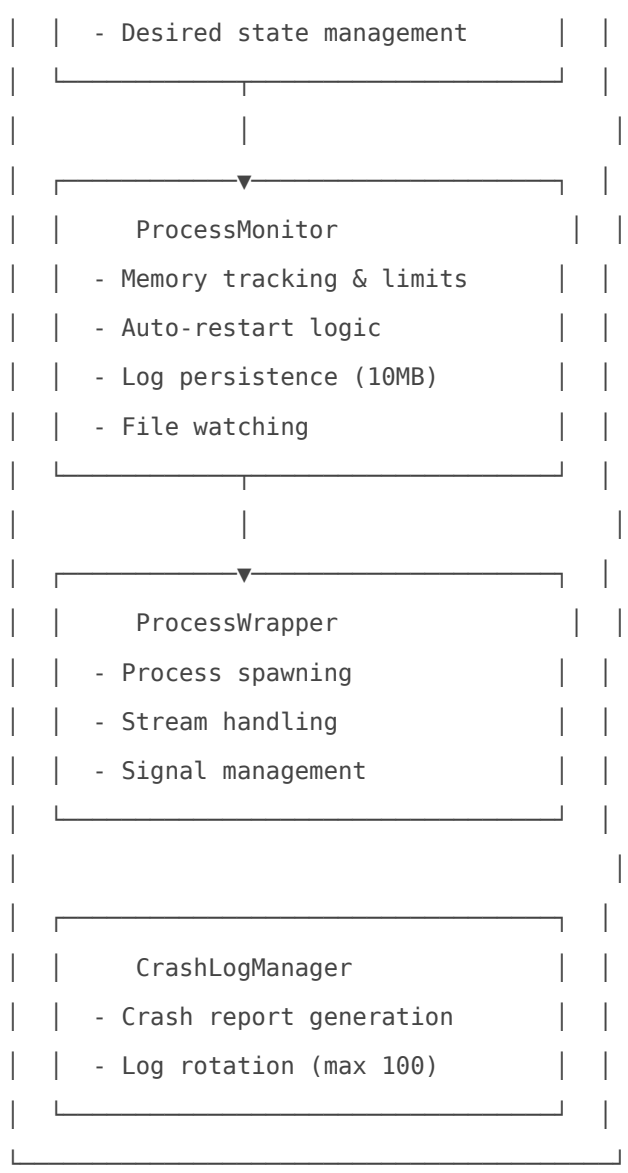
Version Check

```
tspm -v
# tspm CLI: 5.x.y
# Daemon: running v5.x.z (pid 1234)
# Version mismatch detected → optionally refresh the systemd service
```

□□ Architecture

TSPM uses a clean three-tier architecture:

```
┌──────────────────────────────────────────┐
│            CLI Interface                 │
│         (tspm commands)                  │
└──────────────────┬───────────────────────┘
                   │ Unix Socket IPC
┌──────────────────▼───────────────────────┐
│          TSPM Daemon                     │
│     (Background Service)                 │
│  ┌──────────────────────────────────┐    │
│  │     ProcessManager               │    │
│  │  - Configuration persistence     │    │
│  │  - Process lifecycle             │    │
```



Key Components

Component	Role
CLI	Lightweight client that sends commands to daemon via IPC
Daemon	Persistent background service managing all processes
ProcessManager	High-level orchestration, config persistence, state management
ProcessMonitor	Memory limits, auto-restart with backoff, log persistence, file watching
ProcessWrapper	Low-level process lifecycle, stream handling, signal management
CrashLogManager	Detailed crash reports with metadata and log history

□□ Programmatic API

TSPM exposes a typed IPC client for programmatic use:

```
import { TspmIpcClient } from '@git.zone/tspm/client';

const client = new TspmIpcClient();
await client.connect();

// Add a process configuration
const { id } = await client.request('add', {
  config: {
    command: 'node worker.js',
    name: 'background-worker',
    projectDir: process.cwd(),
    memoryLimitBytes: 512 * 1024 * 1024,
    autorestart: true,
  },
});

// Start it
await client.request('startById', { id });

// Get process info
const { processInfo } = await client.request('describe', { id });
console.log(`Status: ${processInfo.status}, Memory: ${processInfo.memory} bytes`);

// Get logs
const { logs } = await client.request('getLogs', { id, limit: 100 });
logs.forEach(log => console.log(`[${log.timestamp}] [${log.type}] ${log.message}`));

// Stop and remove
await client.request('stop', { id });
await client.request('delete', { id });
await client.disconnect();
```

Module Exports

Export Path	Purpose
`@git.zone/tspm`	Main entry point (re-exports client + daemon)
`@git.zone/tspm/client`	IPC client (`TspmIpcClient`, `TspmServiceManager`)
`@git.zone/tspm/daemon`	Daemon entry point (`startDaemon`)
`@git.zone/tspm/protocol`	IPC type definitions

□□ Advanced Features

Restart Backoff & Failure Handling

TSPM handles crashed processes with intelligent backoff:

- **Incremental delay**: Grows linearly from 1s up to 10s for consecutive restarts
- **Failure threshold**: After 10 consecutive failures, the process is marked `errored` and auto-restart stops
- **Auto-reset**: The retry counter resets if no failure occurs for 1 hour
- **Manual recovery**: `tspm restart id:1` always works, even on errored processes

Memory Management

Full process tree memory tracking:

- Discovers all child processes via `ps-tree`
- Calculates combined memory usage across the entire tree
- Gracefully restarts when limit is exceeded (SIGTERM → SIGKILL)
- Prevents memory leaks from taking down production systems

Log Persistence

Smart in-memory log management:

- 10MB ring buffer per process with automatic trimming
- Flushes to disk on stop, restart, or crash
- Reloads persisted logs when process restarts
- Crash logs stored separately with full metadata (exit code, signal, memory, timestamps)

Graceful Shutdown

Multi-stage shutdown for reliability:

1. Send **SIGTERM** for graceful shutdown
2. Wait **5 seconds** for process cleanup
3. Send **SIGKILL** if still alive
4. Clean up **all child processes** in the tree

File Watching

Development-friendly auto-restart:

- Watch specific directories or files
- `node_modules` ignored by default
- Debounced restart on file changes
- Configurable via `--watch-paths`

□□ Debugging

```
# Check daemon status
tspm daemon status

# View process logs
tspm logs name:my-app --lines 200

# Check daemon stderr
tail -f /tmp/daemon-stderr.log

# Force daemon restart
tspm daemon restart
```

Common issues:

Problem	Solution
"Daemon not running"	`tspm daemon start` or `sudo tspm enable`
"Permission denied"	Check socket permissions in `~/.tspm/`
Process won't start	Check logs with `tspm logs <target>`
Memory limit exceeded	Increase with `tspm edit <target>`

□□ Why TSPM?

Feature	TSPM	PM2
TypeScript Native	□ Built in TypeScript	□ JavaScript
Memory Tracking	□ Full process tree	⚠ Main process only
Log Management	□ Smart 10MB buffer	⚠ Can grow unbounded
Architecture	□ Clean 3-tier daemon	□ Monolithic
Dependencies	□ Minimal	□ Heavy
ESM Support	□ Native	⚠ Partial
Crash Reports	□ Detailed with metadata	□ Basic

Perfect For

- □□ **Production Node.js apps** — Reliable process management with memory guards
- □□ **Microservices** — Manage multiple services from a single tool
- □□□ **Development** — File watching and instant auto-restart
- □□ **Workers & Jobs** — Queue workers, cron jobs, background tasks
- □□ **Resource-constrained environments** — Memory limits prevent OOM kills

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 #4
Created 2026-03-28 11:08:52 UTC by foss.global Team
Updated 2026-03-28 12:15:33 UTC by foss.global Team