@git.zone/tsrun

direct execution of TypeScript projects with ts-node

- readme.md for @git.zone/tsrun
- changelog.md for @git.zone/tsrun

readme.md for @git.zone/tsrun

npm version License: MIT TypeScript Node.js

> Run TypeScript files instantly — no build step, no config, no friction ⚡

Execute TypeScript programs on-the-fly with zero configuration. Whether you're writing quick scripts, prototyping ideas, or orchestrating complex multi-project workflows, tsrun gets out of your way and lets you focus on code.

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.

Table of Contents

- What is tsrun?
- Installation
- CLI Usage
- Programmatic API
  - runPath()
  - runCli()
  - spawnPath()
- API Quick Reference
- Features

- Common Use Cases
- Troubleshooting
- License and Legal Information

What is tsrun?

tsrun is a lightweight TypeScript execution tool that lets you run `.ts` files directly — just like running JavaScript with `node`, but for TypeScript. Under the hood, it uses tsx for lightning-fast transpilation, so there's no compilation step, no tsconfig fiddling, and no waiting around.

It also doubles as a **programmatic library** with full process control — spawn TypeScript processes, capture their output, set timeouts, and cancel them with `AbortController`. Perfect for build scripts, task runners, and orchestration tools.

Installation

Global (recommended for CLI usage):

```
pnpm install -g @git.zone/tsrun
```

As a project dependency (for programmatic API):

```
pnpm install @git.zone/tsrun
```

□□ CLI Usage

Run any TypeScript file:

```
tsrun myScript.ts
```

Arguments pass through transparently, just like `node`:

```
tsrun deploy.ts production --verbose --dry-run
```

Your script sees them in `process.argv` as expected:

```
// deploy.ts
const env = process.argv[0];     // "production"
const verbose = process.argv.includes('--verbose');
const dryRun = process.argv.includes('--dry-run');

console.log(`Deploying to ${env}...`);
```

□□ Programmatic API

tsrun exports three functions tailored for different execution needs.

`runPath()` — Simple Execution

Runs a TypeScript file and waits for it to complete. The simplest way to execute scripts programmatically.

```
import { runPath } from '@git.zone/tsrun';

// Run a script (path relative to cwd)
await runPath('./scripts/build.ts');

// Resolve path relative to the calling file
await runPath('./build.ts', import.meta.url);

// Run in a different working directory (spawns a child process)
await runPath('./build.ts', import.meta.url, { cwd: '/path/to/project' });
```

How it works:

- Without `cwd` — executes **in-process** using tsx's ESM loader (fast, zero overhead)
- With `cwd` — spawns an **isolated child process** with the given working directory

`runCli()` — CLI Mode

Runs with `process.argv` integration, as if the script were invoked from the command line. This is what the `tsrun` CLI binary uses internally.

```
import { runCli } from '@git.zone/tsrun';

// Uses process.argv for argument passing
await runCli('./script.ts');

// With custom working directory
await runCli('./script.ts', { cwd: '/path/to/project' });
```

`spawnPath()` — Advanced Process Control

Returns immediately with a process handle, giving you full control over stdio, timeouts, and cancellation.

```
import { spawnPath } from '@git.zone/tsrun';

const proc = spawnPath('./task.ts', import.meta.url, {
  timeout: 30000,                    // Kill after 30s
  cwd: '/path/to/project',
  env: { NODE_ENV: 'production' },   // Merged with process.env
  args: ['--verbose'],               // Extra CLI args
  stdio: 'pipe',                     // Default: capture stdout/stderr
});

// Stream stdout
proc.stdout?.on('data', (chunk) => {
  console.log(chunk.toString());
});

// Wait for exit
const exitCode = await proc.exitCode;
```

AbortController support:

```
const controller = new AbortController();
const proc = spawnPath('./task.ts', import.meta.url, {
  signal: controller.signal,
});

// Cancel from outside
```

```
setTimeout(() => controller.abort(), 5000);

try {
  await proc.exitCode;
} catch (err) {
  console.log('Process was cancelled');
}
```

Graceful termination:

```
const proc = spawnPath('./server.ts', import.meta.url);

// Sends SIGTERM, waits 5s, then SIGKILL if still running
await proc.terminate();
```

API Quick Reference

Function	Execution	Returns	Best For
`runPath()`	In-process (or child with `cwd`)	`Promise<void>`	Simple script execution, sequential workflows
`runCli()`	In-process (or child with `cwd`)	`Promise<void>`	CLI-like invocation with argv integration
`spawnPath()`	Always child process	`ITsrunChildProcess`	Output capture, timeouts, cancellation, parallel tasks

Decision guide:

- □□ **Just run a script?** → `runPath()`
- □□ **Need argv pass-through?** → `runCli()`
- □□ **Need stdout/stderr, timeout, or cancel?** → `spawnPath()`
- ⚡ **Parallel execution across projects?** → `runPath()` with `cwd` or `spawnPath()`

`ITsrunChildProcess` Interface

The object returned by `spawnPath()`:

Property / Method	Type	Description
`childProcess`	`ChildProcess`	Direct access to Node's ChildProcess

Property / Method	Type	Description
`stdout`	`Readable | null`	Stdout stream (`null` if stdio is `'inherit'`)
`stderr`	`Readable | null`	Stderr stream (`null` if stdio is `'inherit'`)
`exitCode`	`Promise<number>`	Resolves with exit code when process ends
`kill(signal?)`	`(signal?: NodeJS.Signals) => boolean`	Send a signal to the process
`terminate()`	`() => Promise<void>`	Graceful shutdown: SIGTERM → 5s → SIGKILL

`ISpawnOptions`

Options for `spawnPath()`:

Option	Type	Default	Description
`cwd`	`string`	`process.cwd()`	Working directory for the child process
`env`	`Record<string, string>`	`{}`	Extra env vars (merged with `process.env`)
`args`	`string[]`	`[]`	Additional CLI arguments
`stdio`	`'pipe' | 'inherit'`	`'pipe'`	Stdio configuration
`timeout`	`number`	—	Auto-kill after N milliseconds
`signal`	`AbortSignal`	—	External cancellation support

Features

 Zero Configuration — Point and shoot. No tsconfig required, no build step, no setup.

⚡ **Lightning Fast** — Powered by tsx (esbuild under the hood) for near-instant TypeScript execution.

 Transparent Arguments — CLI args pass through seamlessly to your scripts via `process.argv`.

 Dual Interface — Use as a CLI tool or import as a library with full TypeScript types.

 Custom Working Directory — Run scripts in isolated child processes with different cwds.

Full Process Control — `spawnPath()` gives you streams, timeouts, cancellation, and graceful shutdown.

Signal Forwarding — SIGINT/SIGTERM/SIGHUP are properly forwarded to child processes.

Common Use Cases

Quick Scripts & Prototyping

```
# Write TypeScript, run it immediately
tsrun seed-database.ts
tsrun generate-report.ts --format csv
tsrun cleanup-temp-files.ts
```

Sequential Build Pipeline

```
import { runPath } from '@git.zone/tsrun';

const steps = ['./lint.ts', './test.ts', './build.ts', './deploy.ts'];

for (const step of steps) {
  console.log(`▶ Running ${step}...`);
  await runPath(step, import.meta.url);
  console.log(`✓ Done`);
}
```

Parallel Multi-Project Builds

```
import { runPath } from '@git.zone/tsrun';

await Promise.all([
  runPath('./build.ts', undefined, { cwd: '/workspace/frontend' }),
  runPath('./build.ts', undefined, { cwd: '/workspace/backend' }),
  runPath('./build.ts', undefined, { cwd: '/workspace/shared' }),
]);
```

Long-Running Tasks with Monitoring

```
import { spawnPath } from '@git.zone/tsrun';

const proc = spawnPath('./data-migration.ts', import.meta.url, {
  timeout: 300000,  // 5 minute max
  env: { LOG_LEVEL: 'verbose' },
});

let lines = 0;
proc.stdout?.on('data', (chunk) => {
  lines++;
  if (lines % 100 === 0) console.log(`Processed ${lines} lines...`);
});

try {
  await proc.exitCode;
  console.log('Migration completed!');
} catch (err) {
  console.error('Migration failed:', err.message);
  process.exit(1);
}
```

Troubleshooting

"Cannot find module" errors

Use `import.meta.url` for path resolution relative to the calling file:

```
// ❌ Relative to cwd — fragile
await runPath('./script.ts');

// ✅ Relative to current file — reliable
await runPath('./script.ts', import.meta.url);
```

Process hangs

When using `spawnPath()`, always await the `exitCode` promise:

```
const proc = spawnPath('./script.ts', import.meta.url);
await proc.exitCode; // Don't forget this!
```

Timeout only works with `spawnPath()`

`runPath()` executes in-process and doesn't support timeouts. Use `spawnPath()` instead:

```
const proc = spawnPath('./script.ts', import.meta.url, { timeout: 5000 });
await proc.exitCode;
```

`tsrun: command not found`

Install globally or use `npx`:

```
pnpm install -g @git.zone/tsrun
# or
npx @git.zone/tsrun myScript.ts
```

License and Legal Information

Trademarks

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.

changelog.md for @git.zone/tsrun

2026-03-24 - 2.0.2 - fix(build)

migrate package metadata to smartconfig and refresh build configuration

- replace npmextra.json with .smartconfig.json and update published package files
- remove the web build flag from the build script
- refresh dependency versions and remove the unused node-fetch dev dependency
- rewrite the README to reflect current CLI, API, and Node.js support

2025-12-13 - 2.0.1 - fix(cli)

Align package scope to @git.zone, bump dependency versions and remove obsolete pnpm workspace setting

- Update runtime import in cli.ts.js from @gitzone/tsrun to @git.zone/tsrun
- Change npm package name in npmextra.json to @git.zone/tsrun
- Bump devDependencies and dependencies in package.json (@git.zone/tsbuild -> ^3.1.2, @push.rocks/smartcli -> ^4.0.19, @types/node -> ^25.0.1, @push.rocks/smartfile -> ^13.1.0, tsx -> ^4.21.0)
- Remove onlyBuiltDependencies entry from pnpm-workspace.yaml
- Ensure commitinfo metadata (ts/00_commitinfo_data.ts) and package.json remain aligned with @git.zone/tsrun

2025-11-17 - 2.0.0 - BREAKING CHANGE(tsconfig)

Remove experimentalDecorators and useDefineForClassFields from tsconfig.json

- tsconfig.json: removed compilerOptions.experimentalDecorators — decorator support is no longer enabled by default. Projects using decorators must enable experimentalDecorators in their own tsconfig.
- tsconfig.json: removed compilerOptions.useDefineForClassFields — class field emit will follow TypeScript defaults, which may change runtime semantics for some classes.
- This may break consumers relying on the previous compiler options; bumping the major version to reflect the potential breaking change.

2025-10-17 - 1.6.2 - fix(ts/index)

Use cli.js as the spawned CLI entry point instead of cli.child.js

- Replace references to ../cli.child.js with ../cli.js in ts/index.ts (runInChildProcess and spawnPath) to ensure child processes spawn the correct CLI entry point.
- This change fixes child process spawning failures caused by referencing a non-existent cli.child.js file.
- Add local .claude/settings.local.json (local runner/editor permissions configuration).

2025-10-17 - 1.6.1 - fix(plugins)

Export child_process.spawn from plugins and use plugins.spawn in spawnPath to remove direct require and unify process spawning

- Exported spawn from ts/plugins.ts so native child_process.spawn is available via the plugins module
- Removed require('child_process') from ts/index.ts and switched to plugins.spawn when spawning child processes in spawnPath
- No public API changes; this unifies internal imports and fixes inconsistent spawn usage that could cause runtime issues

2025-10-17 - 1.6.0 - feat(core)

Add spawnPath child-process API with timeout/abort/terminate support, export native types, and expand README

- Implement spawnPath(filePath, fromFileUrl?, options?) in ts/index.ts producing an ITsrunChildProcess with childProcess, stdout, stderr, exitCode, kill() and terminate()
- Introduce ISpawnOptions (cwd, env, args, stdio, timeout, signal) and ITsrunChildProcess interfaces for robust process control

- Handle timeouts (auto SIGTERM), AbortSignal cancellation, and graceful terminate() (SIGTERM then SIGKILL after 5s)
- Export Node types ChildProcess and Readable from ts/plugins.ts for improved typings
- Greatly expand README: add badges, table of contents, detailed API docs and examples for runPath, runCli and spawnPath, and troubleshooting guidance
- Add local .claude/settings.local.json (environment/settings file)

2025-10-16 - 1.5.0 - feat(core)

Add cwd option and child-process execution for custom working directory; implement signal-forwarding child runner; update docs and bump package version to 1.4.0

- Introduce IRunOptions with cwd support to runPath/runCli
- When cwd is provided, runCli now spawns a child process (runInChildProcess) to execute the script in the specified working directory
- runInChildProcess preserves node execArgv, inherits env and stdio, forwards signals (SIGINT, SIGTERM, SIGHUP) and propagates child exit codes/signals
- Update README with documentation and examples for running scripts with a custom working directory and parallel execution
- Bump package version to 1.4.0

2025-10-13 - 1.3.4 - fix(docs)

Update README with expanded docs and examples; add pnpm and CI tooling configs

- Rewrite and expand README: clearer intro, installation, CLI and programmatic usage examples, features and examples, and updated package/project links.
- Add packageManager entry to package.json to record pnpm version/hash (pnpm@10.18.1+sha512...).
- Add pnpm-workspace.yaml with onlyBuiltDependencies for esbuild.
- Add .serena/project.yml and .serena/.gitignore for project metadata and Serena tooling configuration.
- Add .claude/settings.local.json to configure local agent permissions.
- No functional TypeScript source changes in this commit (runtime implementations remain as placeholders).

2024-10-27 - 1.3.3 - fix(core)

removed unused import statement in ts/plugins.ts

- Cleanup: Removed an unused import statement for tsImport from tsx/esm/api

2024-10-27 - 1.3.2 - fix(core)

Replace ts-node with tsx for module handling

- Removed ts-node and its loader, using tsx for module imports
- Simplified import logic by replacing tsx register API call
- Updated dependencies in package.json by removing ts-node and typescript

2024-10-27 - 1.3.1 - fix(core)

Add console.log to show ts-node options in use

- Added a console log statement in the ts/loader.ts to display the default ts-node options being used.

2024-10-27 - 1.3.0 - feat(ci)

Add Gitea workflows for build and release.

- Added .gitea/workflows/default_nottags.yaml for non-tag events.
- Added .gitea/workflows/default_tags.yaml for tag-based events.
- Updated build scripts in package.json to use pnpm.
- Refactored imports and exports in TypeScript source for better modularity.

2024-06-24 - 1.2.49 - fix(core)

Minor maintenance updates with version bump

2024-06-24 - 1.2.48 - fix(dependencies)

Update TypeScript dependency version

- Updated TypeScript from version 5.4.5 to 5.5.2 to include latest features and fixes.

2024-06-24 - 1.2.47 - fix(core)

Remove GitLab CI configuration and update dependencies

- Removed the .gitlab-ci.yml file.
- Updated @git.zone/tsbuild from ^2.1.69 to ^2.1.80.
- Updated @push.rocks/smartcli from ^4.0.8 to ^4.0.11.
- Updated @types/node from ^20.5.6 to ^20.14.8.
- Updated @push.rocks/smartfile from ^10.0.30 to ^11.0.21.
- Updated @push.rocks/smartshell from ^3.0.3 to ^3.0.5.
- Updated ts-node from ^10.9.1 to ^10.9.2.
- Updated typescript from 5.1.6 to 5.4.5.

2023-08-26 - 1.2.44 to 1.2.46 - Core Updates and Fixes

Several internal updates and fixes to the core functionality.

- Continuous core updates for improved performance and bug fixes

2023-07-13 - 1.2.42 to 1.2.44 - Core Updates and Fixes

Addressing minor bugs and improving core operations.

- Several bug fixes to enhance stability

2023-06-03 - 1.2.39 to 1.2.42 - Core and Stability Improvements

Focus on refining core processes and fixing issues.

- Significant fixes for better core performance

2022-10-12 - 1.2.37 to 1.2.39 - Core Updates

Series of bug fixes to maintain core functionality.

- Addressed minor bugs in core areas

2022-06-02 - 1.2.34 to 1.2.37 - Core Maintenance

Routine updates to address and fix core issues.

- Bug fixes for better core stability

2022-03-11 to 2022-03-13 - 1.2.18 to 1.2.33 - Core Enhancements

Multiple updates focusing on resolving core issues and maintaining performance.

- Continuous improvements and fixes in the core functions

2021-10-06 - 1.2.18 to 1.2.19 - Core Updates

Minor fixes to keep up with core performance.

- Addressed core issues for improved functionality

2021-06-23 to 2021-06-24 - 1.2.12 to 1.2.17 - Core Improvements

Series of updates addressing core functionality.

- Enhanced core features and fixed known bugs

2020-06-01 - 1.2.8 to 1.2.11 - Core Stability

Fixes focusing on ensuring core stability.

- Stability improvements in core components

2019-07-17 - 1.2.6 to 1.2.7 - Core Updates

Minor fixes targeting the core framework.

- Adjusted core components to enhance performance

2019-04-08 to 2019-07-17 - 1.2.2 to 1.2.6 - ES2017 Compatibility and Core Fixes

Updated core to support ES2017 and addressed various core issues.

- Updated environment compatibility
- Fixed several core issues

2018-12-06 - 1.1.13 to 1.1.17 - Core Fixes

Routine updates addressing core bugs and issues.

- Multiple fixes to core functionality

2018-08-08 - 1.1.11 to 1.1.12 - Dependency Updates

Updated dependencies critical for core performance.

- Enhanced dependencies for better performance

2018-07-13 - 1.1.4 to 1.1.10 - Various Updates

Multiple updates focused on documentation, dependencies, and core functions.

- Fixed documentation
- Updated core
- Removed obsolete dependencies

2018-06-30 to 2018-07-03 - 1.0.8 to 1.1.3 - Feature and Core Fixes

Introduced new features and addressed core issues.

- Enabled new CLI options
- Resolved core issues for better functionality

2018-06-25 - 1.0.4 to 1.0.7 - Core and Security Improvements

Made core improvements and updated security features.

- Added security files
- Fixed core argument parsing issues

2018-06-04 - 1.0.1 to 1.0.3 - Initial Fixes and Improvements

Initial setup and fixes to existing issues.

- Moved dependencies to dev
- Fixed package details