readme.md for @git.zone/tspublish

> **Effortlessly publish multiple TypeScript packages from your monorepo**

npm version License: MIT

□□ What is tspublish?

`@git.zone/tspublish` is a powerful CLI tool and library for managing and publishing multiple TypeScript packages from a monorepo. It automates the tedious parts of package publishing — discovery, dependency resolution, building, version validation, and multi-registry publishing — while giving you full control over the process. Whether you're maintaining a suite of microservices, a component library, or any collection of related packages, tspublish makes your life dramatically easier.

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.

□ Key Features

- □□ **Automatic Package Discovery** — Scans your monorepo for publishable `ts*` directories
- □□ **Beautiful CLI Output** — Color-coded logging with progress bars and status indicators
- □□ **Version Collision Detection** — Prevents accidental overwrites by checking the registry first

- □□ **Build Integration** — Automatically compiles TypeScript before publishing via `@git.zone/tsbuild`
- □□ **Smart Dependency Management** — Inherits dependency versions from your monorepo's `package.json`
- □□ **Multi-Registry Support** — Publish to npm, GitHub Packages, Gitea, or private registries
- □□ **Base Registry Inheritance** — Use `useBase` / `extendBase` to inherit registries from `.smartconfig.json`
- ⚡ **CLI Binary Support** — Automatically generates `cli.js` wrappers for publishable CLI packages
- □□ **Clean Builds** — Creates isolated `dist_publish_*` directories for each package

□□ Installation

```
# Using pnpm (recommended)
pnpm add -D @git.zone/tspublish

# Global installation for CLI usage
pnpm add -g @git.zone/tspublish
```

□□ Quick Start

1□ Structure Your Monorepo

Organize your packages using directories that start with `ts`:

```
my-awesome-monorepo/
├── package.json            # Main monorepo package.json (version is inherited)
├── tsconfig.json           # Shared TypeScript config
├── .smartconfig.json       # Optional: base registry configuration
├── ts_core/                # Core package
│   ├── index.ts            # Entry point
│   ├── readme.md           # Package-specific documentation
│   └── tspublish.json      # Publishing configuration
├── ts_utils/               # Utilities package
│   ├── index.ts
│   ├── readme.md
```

```
│   └── tspublish.json
└── ts_cli/                # CLI package
    ├── index.ts
    ├── readme.md
    └── tspublish.json
```

2️⃣ Configure Each Package

Create a `tspublish.json` in each publishable directory:

```
{
  "name": "@myorg/core",
  "order": 1,
  "dependencies": [
    "@push.rocks/smartpromise",
    "@push.rocks/smartfile"
  ],
  "registries": [
    "registry.npmjs.org:public"
  ],
  "bin": []
}
```

Configuration Options

Field	Type	Description
`name`	`string`	The published package name (e.g., `@myorg/core`)
`order`	`number`	Build order for interdependent packages (lower builds first)
`dependencies`	`string[]`	Dependencies to include — versions are resolved from the monorepo's `package.json`
`registries`	`string[]`	Target registries with access level (format: `"url:accessLevel"`)
`bin`	`string[]`	CLI executable names (generates a `cli.js` wrapper automatically)

3️⃣ Publish

```
# From your monorepo root
npx tspublish
```

That's it! tspublish will discover all `ts*` directories containing `tspublish.json`, build them in order, validate versions against the registry, and publish.

□□ Advanced Usage

Registry Configuration

TSPublish offers three approaches for configuring target registries:

1. Explicit Registries

Define specific registries directly in `tspublish.json`:

```
{
  "registries": [
    "registry.npmjs.org:public",
    "npm.pkg.github.com:private"
  ]
}
```

The format is `"registryUrl:accessLevel"` where `accessLevel` is `public` or `private`.

2. Use Base Configuration (`useBase`)

Inherit registries from your project's `.smartconfig.json` (managed by `@git.zone/cli`):

```
{
  "registries": ["useBase"]
}
```

This reads from `.smartconfig.json` at the key `@git.zone/cli.release.registries`.

3. Extend Base Configuration (`extendBase`)

Start with base registries and add or remove specific ones:

```
{
  "registries": [
    "extendBase",
    "custom-registry.example.com:public",
    "-https://registry.npmjs.org"
  ]
}
```

The `-` prefix excludes a registry from the base configuration. All other entries (besides `extendBase`) are added.

Empty Registries

If `registries` is an empty array `[]`, the package will be built but **not published** — useful for internal-only packages that other packages depend on.

Build Order Management

When packages depend on each other, use the `order` field to control build sequence:

```
// ts_core/tspublish.json — builds first
{
  "name": "@myorg/core",
  "order": 1,
  "dependencies": [],
  "registries": ["useBase"],
  "bin": []
}

// ts_utils/tspublish.json — builds second, depends on core
{
  "name": "@myorg/utils",
  "order": 2,
  "dependencies": ["@myorg/core"],
  "registries": ["useBase"],
  "bin": []
}
```

CLI Binary Packages

For packages that ship CLI tools, specify the binary names in the `bin` array:

```
{
  "name": "@myorg/cli",
  "order": 3,
  "dependencies": ["commander", "@myorg/core"],
  "registries": ["registry.npmjs.org:public"],
  "bin": ["my-cli", "my-tool"]
}
```

TSPublish will:

1. Fetch the standard `cli.js` template from the `@git.zone/cli` assets repository
2. Adjust the import path to point to the correct `dist_*` folder
3. Configure the `bin` field in the generated `package.json`

Programmatic Usage

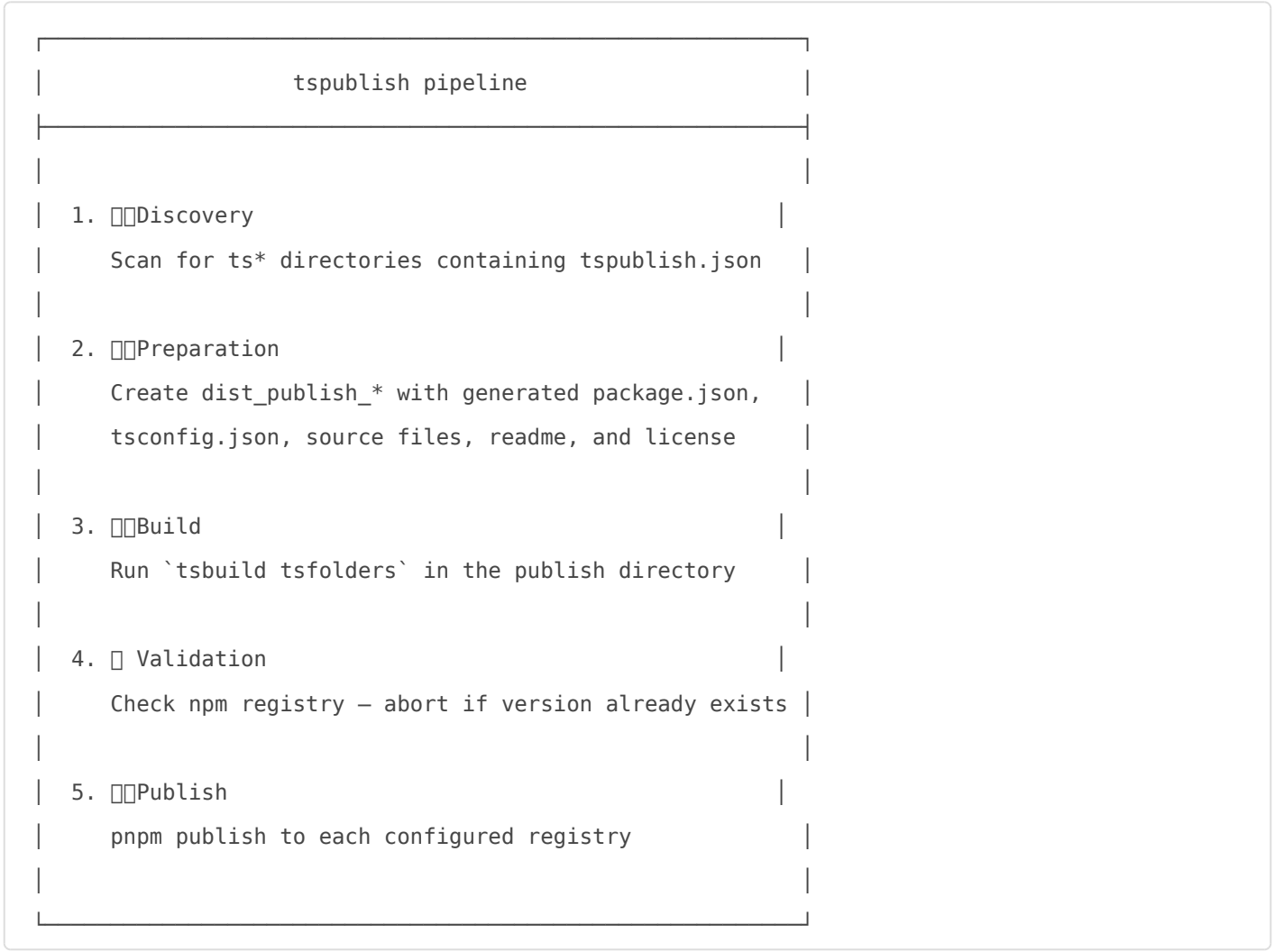
```
import { TsPublish } from '@git.zone/tspublish';

const publisher = new TsPublish();
await publisher.publish(process.cwd());
```

Custom Publishing Pipeline

```
import { TsPublish, PublishModule } from '@git.zone/tspublish';

const publisher = new TsPublish();
const modules = await publisher.getModuleSubDirs('./my-monorepo');

for (const [name, config] of Object.entries(modules)) {
  const mod = new PublishModule(publisher, {
    monoRepoDir: './my-monorepo',
    packageSubFolder: name,
  });

  await mod.init();                    // Resolve deps, validate version
  await mod.createPublishModuleDir(); // Scaffold dist_publish_* directory
  await mod.build();                   // Compile TypeScript
```

```
  await mod.publish();               // Publish to registries
}
```

□□ How It Works



For each discovered module, tspublish:

1. **Discovers** all directories starting with `ts` that contain a `tspublish.json`
2. **Resolves dependencies** from the monorepo's `package.json`, using the monorepo version for packages not found in `dependencies`
3. **Creates an isolated publish directory** (`dist_publish_<folder>`) with a generated `package.json`, `tsconfig.json`, source code copy, readme, and license
4. **Builds** the package using `pnpm run build` (which calls `tsbuild tsfolders`)
5. **Validates** against the target registry — if the name+version already exists, it exits with an error
6. **Publishes** to each configured registry via `pnpm publish`

□□ API Reference

TsPublish

```
class TsPublish {
  /** Publish all discovered modules in a monorepo directory */
  async publish(monorepoDirPath: string): Promise<void>;

  /** Discover and return all publishable modules with their tspublish.json configs */
  async getModuleSubDirs(dirPath: string): Promise<Record<string, ITsPublishJson>>;
}
```

PublishModule

```
class PublishModule {
  /** Initialize: resolve dependencies, validate version against registry */
  async init(): Promise<void>;

  /** Create the dist_publish_* directory with all necessary files */
  async createPublishModuleDir(): Promise<void>;

  /** Build the TypeScript package */
  async build(): Promise<void>;

  /** Publish to all configured registries */
  async publish(): Promise<void>;
}
```

ITsPublishJson

```
interface ITsPublishJson {
  name: string;           // Published package name
  order: number;          // Build sequence (lower = earlier)
  dependencies: string[]; // Dependencies to include from monorepo
  registries: string[];   // Target registries ("url:accessLevel", "useBase", or "extendBase")
```

```
  bin: string[];          // CLI binary names
}
```

GiteaAssets

```
class GiteaAssets {
  constructor(options: { giteaBaseUrl: string; token?: string });

  /** Fetch files from a Gitea repository directory */
  async getFiles(owner: string, repo: string, directory: string, branch?: string):
Promise<IRepoFile[]>;

  /** Get the standard cli.js entry file template */
  async getBinCliEntryFile(): Promise<IRepoFile>;
}
```

□□ Troubleshooting

Problem	Solution
"Package X already exists with version Y"	Bump the version in your monorepo's `package.json`
No publish modules found	Ensure directories start with `ts` and contain a valid `tspublish.json`
Build failures	Check TypeScript errors — tspublish runs `tsbuild tsfolders` in the generated directory
useBase/extendBase error	Ensure `.smartconfig.json` has registries at `@git.zone/cli.release.registries`
Missing dependency versions	Add the dependency to your monorepo's `package.json` `dependencies` field

License and Legal Information

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 12:15:34 UTC by foss.global Team