@git.zone/tsbuild

Documentation for @git.zone/tsbuild

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

readme.md for @git.zone/tsbuild

A powerful, modern TypeScript build tool with smart defaults, full tsconfig.json support, automatic output directory management, and cross-module import path rewriting.

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.

Install

```
# Using pnpm (recommended)
pnpm install @git.zone/tsbuild --save-dev

# Using npm
npm install @git.zone/tsbuild --save-dev
```

Why tsbuild?

Feature	Description
Smart tsconfig.json Integration	Respects all your compiler options with intelligent merging
Protected Defaults	Critical build settings are safeguarded while staying flexible
Zero Config	Works perfectly without tsconfig.json
Glob Pattern Support	Compile multiple directories with a single command

Feature	Description
Dependency-Aware	Automatically orders compilation based on module dependencies
Type Checking	Validate code without emitting files
Clean Builds	Automatically clears output directories before compilation
Auto-Unpack	Flattens nested output directories automatically
Import Path Rewriting	Rewrites cross-module imports to point at compiled output
CI/CD Ready	JSON output mode and proper exit codes
Modern Defaults	ESNext, NodeNext modules, decorators out of the box

Quick Start

CLI Usage

Compile your TypeScript project:

```
npx tsbuild
```

Compiles `./ts/**/*.ts` to `./dist_ts/`

Custom directories:

```
npx tsbuild custom src utils
```

Compiles:

- `./src/**/*.ts` -> `./dist_src/`
- `./utils/**/*.ts` -> `./dist_utils/`

Auto-discover and compile in dependency order:

```
npx tsbuild tsfolders
```

Finds all `ts_*` folders and compiles them respecting dependencies.

Programmatic Usage

Basic compilation:

```
import { TsCompiler } from '@git.zone/tsbuild';

const compiler = new TsCompiler();
await compiler.compileFilesOrThrow([
  './src/index.ts',
  './src/utils.ts'
], { outDir: './dist' });
```

Production-ready with error tracking (recommended):

```
import { TsCompiler } from '@git.zone/tsbuild';

const compiler = new TsCompiler();
const result = await compiler.compileFiles([
  './src/index.ts',
  './src/utils.ts'
], { outDir: './dist' });

if (result.errorSummary.totalErrors > 0) {
  console.error(`Compilation failed with ${result.errorSummary.totalErrors} errors`);
  process.exit(1);
}

console.log(`Compiled ${result.emittedFiles.length} files successfully!`);
```

Glob pattern compilation:

```
import { TsCompiler } from '@git.zone/tsbuild';

const compiler = new TsCompiler();
await compiler.compileGlob({
  './ts/**/*.ts': './dist_ts',
  './ts_web/**/*.ts': './dist_web'
});
```

CLI Commands

1. Default Build

```
npx tsbuild [options]
```

Compiles all TypeScript files from `./ts/` to `./dist_ts/`

Options:

Flag	Description
`--skiplibcheck`	Skip type checking of declaration files
`--confirmskiplibcheck`	Skip lib check with extended warning (5s pause)
`--disallowimplicitany`	Disallow implicit `any` types
`--commonjs`	Use CommonJS instead of ESNext modules
`--json`	Output results as JSON (for CI/CD)
`--quiet`	Suppress console output

Examples:

```
# Standard build
npx tsbuild

# Build with JSON output for CI
npx tsbuild --json --quiet

# CommonJS build
npx tsbuild --commonjs

# Strict mode
npx tsbuild --disallowimplicitany
```

2. Custom Directories

```
npx tsbuild custom <dir1> <dir2> ... [options]
```

Compile specific directories to their corresponding `dist_` folders.

```
# Compile src and utils
npx tsbuild custom src utils
```

```
# Creates: ./dist_src/ and ./dist_utils/

# Multiple directories with options
npx tsbuild custom api models services --commonjs
```

3. TSFolders (Dependency-Aware)

```
npx tsbuild tsfolders [options]
```

Automatically discovers and compiles all `ts_*` folders in dependency order:

1. Prioritizes `ts_interfaces` first (if no tspublish.json)
2. Prioritizes `ts_shared` second (if no tspublish.json)
3. Reads `tspublish.json` in each folder for `order` property
4. Compiles in correct sequence

Example output:

```
TypeScript Folder Compilation Plan (5 folders)
  1. ts_interfaces
  2. ts_shared
  3. ts_core
  4. ts_utils
  5. ts_modules
```

4. Emit Check

```
npx tsbuild emitcheck <file_or_pattern> [more...] [options]
```

Validates TypeScript files can be compiled without actually emitting them.

```
# Check specific files
npx tsbuild emitcheck src/main.ts src/utils.ts

# Check with glob patterns
npx tsbuild emitcheck "src/**/*.ts" "test/**/*.ts"
```

Exit codes:

- `0` - All files can be emitted

- `1` - One or more files have errors

5. Type Check

```
npx tsbuild check [pattern] [more...] [options]
```

Performs type checking without emitting files.

With arguments: Check specified files/patterns

```
npx tsbuild check "ts/**/*.ts"
npx tsbuild check "src/**/*.ts" "test/**/*.ts"
```

Without arguments: Two-phase default check

1. Phase 1: Type check `ts/**/*` (strict, includes .d.ts)
2. Phase 2: Type check `test/**/*` (relaxed, skipLibCheck: true)

```
npx tsbuild check
# Running default type checking sequence...
# Checking ts/**/* files...
# Checking test/**/* files with --skiplibcheck...
# All default type checks passed!
```

API Reference

TsCompiler Class

The main class for TypeScript compilation.

```
import { TsCompiler } from '@git.zone/tsbuild';


const compiler = new TsCompiler(cwd?: string, argvArg?: any);
```

Constructor Parameters:

- `cwd` - Working directory (defaults to `process.cwd()`)
- `argvArg` - CLI arguments object for flags like `--skiplibcheck`, `--quiet`, etc.

compileFiles(fileNames, customOptions?, taskInfo?)

Compile files with error tracking. Returns result instead of throwing.

```
const result = await compiler.compileFiles(
  ['./src/index.ts', './src/utils.ts'],
  { outDir: './dist' }
);

console.log(`Emitted: ${result.emittedFiles.length} files`);
console.log(`Errors: ${result.errorSummary.totalErrors}`);
```

Returns: `Promise<ICompileResult>`

```
interface ICompileResult {
  emittedFiles: string[];
  errorSummary: IErrorSummary;
}

interface IErrorSummary {
  errorsByFile: Record<string, Diagnostic[]>;
  generalErrors: Diagnostic[];
  totalErrors: number;
  totalFiles: number;
}
```

compileFilesOrThrow(fileNames, customOptions?)

Compile files and throw on error. For simple scripts.

```
try {
  const emittedFiles = await compiler.compileFilesOrThrow(
    ['./src/index.ts'],
    { outDir: './dist' }
  );
  console.log('Compiled:', emittedFiles);
} catch (error) {
  console.error('Compilation failed!');
  process.exit(1);
}
```

Returns: `Promise<string[]>` - Array of emitted file paths

compileGlob(globPatterns, customOptions?)

Compile multiple glob patterns to different destinations. Automatically clears output directories before compilation, unpacks nested output, and rewrites cross-module import paths.

```
const result = await compiler.compileGlob({
  './ts/**/*.ts': './dist_ts',
  './ts_web/**/*.ts': './dist_web',
  './ts_node/**/*.ts': './dist_node'
});
```

Returns: `Promise<ICompileResult>`

checkTypes(fileNames, customOptions?)

Type check files without emitting. Fast validation.

```
const success = await compiler.checkTypes(['./src/**/*.ts']);

if (!success) {
  console.error('Type errors found!');
  process.exit(1);
}
```

Returns: `Promise<boolean>` - `true` if no errors

checkEmit(fileNames, customOptions?)

Validate files can be emitted without actually emitting.

```
const canEmit = await compiler.checkEmit(['./src/index.ts']);

if (!canEmit) {
  console.error('Cannot emit these files!');
}
```

Returns: `Promise<boolean>` - `true` if can emit

createOptions(customOptions?)

Get merged compiler options (useful for debugging).

```
const options = compiler.createOptions({ strict: true });
console.log(options); // Shows merged options
```

Supporting Classes

TsConfig

TypeScript configuration management.

```
import { TsConfig } from '@git.zone/tsbuild';

const config = new TsConfig(process.cwd());
const options = config.merge({ target: 'ES2022' }, argvArg);
```

TsPublishConfig

Reads `tspublish.json` for module configuration.

```
import { TsPublishConfig } from '@git.zone/tsbuild';

const pubConfig = new TsPublishConfig('./ts_core');
console.log(pubConfig.shouldUnpack); // true/false
console.log(pubConfig.order);        // number or undefined
```

TsUnpacker

Flattens nested TypeScript output directories.

```
import { TsUnpacker } from '@git.zone/tsbuild';

const unpacker = new TsUnpacker('ts_core', './dist_ts_core');
await unpacker.unpack();
```

TsPathRewriter

Rewrites cross-module import paths in compiled output files. When TypeScript compiles files that import from sibling directories (e.g., `../ts_shared/helper.js`), TsPathRewriter rewrites those paths to reference the compiled output directories (`../dist_ts_shared/helper.js`).

```
import { TsPathRewriter } from '@git.zone/tsbuild';

// Auto-detect all ts_* folders in the project
const rewriter = await TsPathRewriter.fromProjectDirectory(process.cwd());
const filesModified = await rewriter.rewriteDirectory('./dist_ts_core');

// Or from explicit glob patterns
const rewriter2 = TsPathRewriter.fromGlobPatterns({
  './ts_core/**/*.ts': './dist_ts_core',
  './ts_shared/**/*.ts': './dist_ts_shared',
});
```

FsHelpers

Static filesystem utilities.

```
import { FsHelpers } from '@git.zone/tsbuild';

const files = await FsHelpers.listFilesWithGlob('./', 'ts/**/*.ts');
const exists = await FsHelpers.fileExists('./tsconfig.json');
const dirExists = await FsHelpers.directoryExists('./ts');
```

TsBuildCli

CLI command handler. Used internally by the CLI.

```
import { TsBuildCli, runCli } from '@git.zone/tsbuild';

// Run the CLI
runCli();

// Or with custom working directory
const cli = new TsBuildCli('/path/to/project');
cli.run();
```

Configuration

tsconfig.json Support

tsbuild fully supports all compiler options from `tsconfig.json`. Your project configuration is respected and intelligently merged.

```
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "NodeNext",
    "moduleResolution": "NodeNext",
    "strict": true,
    "esModuleInterop": true,
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true,
    "verbatimModuleSyntax": true
  }
}
```

Configuration Priority (5 Levels)

When multiple configuration sources exist, they merge in this order (later overrides earlier):

Priority	Source	Description
1	**Default Options**	tsbuild's sensible defaults
2	**tsconfig.json**	All options from your tsconfig.json (if present)
3	**Protected Defaults**	Critical options for build integrity
4	**Programmatic Options**	Options passed to API functions
5	**CLI Flags**	Command-line arguments (highest priority)

Protected Options

These options cannot be overridden by tsconfig.json alone (but can be overridden programmatically or via CLI):

Option	Value	Reason

`outDir`	`'dist_ts/'`	Required for automatic path transformations
`noEmitOnError`	`true`	Prevents broken builds from being emitted
`declaration`	`true`	Ensures `.d.ts` files for library consumers
`inlineSourceMap`	`true`	Consistent debugging experience

Default Compiler Options

When no tsconfig.json exists:

```
{
  declaration: true,              // Generate .d.ts files
  inlineSourceMap: true,          // Debug-friendly
  noEmitOnError: true,            // Fail-fast on errors
  outDir: 'dist_ts/',             // Output directory
  module: 'NodeNext',             // Modern Node.js modules
  target: 'ESNext',               // Latest JavaScript
  moduleResolution: 'NodeNext',
  noImplicitAny: false,           // Flexible for quick development
  esModuleInterop: true,          // CJS/ESM interop
  verbatimModuleSyntax: true      // Explicit imports/exports
}
```

Path Transformation

tsbuild automatically transforms path mappings:

tsconfig.json:

```
{
  "compilerOptions": {
    "paths": {
      "@models/*": ["./ts_models/*"]
    }
  }
}
```

Automatic transformation:

```
./ts_models/* -> ./dist_ts_models/*
```

Features

Clean Builds

Output directories are automatically cleared before compilation:

```
Clearing output directory: ./dist_ts
Compiling 14 files from ./ts/**/*.ts
```

This ensures no stale files remain from previous builds.

Import Path Rewriting

When working with multi-module projects (multiple `ts_*` folders), TypeScript compiles import paths relative to source directories. After compilation and unpacking, these paths would be wrong because the directory structure changes.

tsbuild automatically detects all `ts_*` folders in the project and rewrites import paths in the compiled output:

```
# Source code
import { helper } from '../ts_shared/helper.js';


# Compiled output (after rewriting)
import { helper } from '../dist_ts_shared/helper.js';
```

This works for ES module imports, dynamic `import()` calls, and CommonJS `require()` statements. The rewriting happens automatically as part of `compileGlob()`.

Monorepo Support with tspublish

tsbuild is designed to work seamlessly with @git.zone/tspublish for monorepo workflows. This enables building and publishing multiple packages from a single repository.

Directory Structure

```
my-project/
  ts/                   # Main package source
  ts_interfaces/        # Shared interfaces (order: 1)
  ts_shared/            # Shared utilities (order: 2)
  ts_core/              # Core logic (order: 3)
  ts_web/               # Web-specific code (order: 4)
  ts_node/              # Node-specific code (order: 5)
```

Each `ts_*` folder can contain its own `tspublish.json` to configure compilation and publishing behavior.

tspublish.json Configuration

Create a `tspublish.json` in each `ts_*` folder:

```
{
  "name": "@myorg/core",
  "order": 3,
  "unpack": true,
  "dependencies": ["ts_interfaces", "ts_shared"]
}
```

Option	Type	Default	Description
`name`	string	--	Package name for publishing
`order`	number	`Infinity`	Build sequence (lower builds first)
`unpack`	boolean	`true`	Flatten nested output directories
`dependencies`	string[]	--	Monorepo dependencies to bundle

Build Order

The `tsfolders` command respects the `order` property:

```
npx tsbuild tsfolders
```

Default ordering (without tspublish.json):

1. `ts_interfaces` - always first (shared types)
2. `ts_shared` - always second (shared utilities)
3. Other folders sorted by `order` property
4. Folders without `order` are built last

Auto-Unpack

When TypeScript compiles files that import from sibling directories, it creates nested output:

```
dist_ts_core/
  ts_core/        <-- nested output
  ts_shared/      <-- pulled-in dependency
```

tsbuild automatically flattens this to:

```
dist_ts_core/
  index.js        <-- flat, clean structure
```

This is especially important for monorepos where packages import from each other.

Control via tspublish.json:

```
{
  "unpack": true
}
```

- `"unpack": true` (default) - Flatten nested directories after compilation
- `"unpack": false` - Keep original nested structure

What gets unpacked:

- The source folder's contents (`ts_core/`) are moved to the root of `dist_ts_core/`
- Sibling folders (`ts_shared/`) that were pulled in are removed (they have their own dist)

Decorator Support

tsbuild supports both **TC39 standard decorators** (recommended) and legacy experimental decorators.

TC39 Standard Decorators (Preferred)

We strongly recommend using TC39 standard decorators for all new code. They are the official JavaScript standard and provide better semantics:

```
// TC39 standard decorator
function log(target: any, context: ClassMethodDecoratorContext) {
  return function (...args: any[]) {
    console.log(`Calling ${String(context.name)}`);
    return target.apply(this, args);
  };
}

class UserService {
  @log
  getUser(id: string) {
    return { id, name: 'John' };
  }
}
```

Legacy Decorators (Backwards Compatibility)

For existing projects using frameworks that still require experimental decorators:

```
{
  "compilerOptions": {
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  }
}
```

Structured Logging

tsbuild uses a 4-level visual hierarchy for console output, making it easy to follow compilation progress:

- **HEADER** - Top-level section start (emoji + bold text + separator)
- **STEP** - Major action within a section (emoji + text)
- **DETAIL** - Supplementary info under a step (indented)
- **SUCCESS/ERROR/WARN** - Outcome indicators with color coding

All output uses ANSI color codes for terminal readability. Use `--quiet` to suppress output or `--json` for machine-readable format.

CI/CD Integration

GitHub Actions

```
name: Build
on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: '22'

      - run: npm install
      - run: npx tsbuild
```

JSON Output

```
npx tsbuild --json --quiet
```

```
{
  "success": true,
  "totals": {
    "errors": 0,
    "filesWithErrors": 0,
    "tasks": 1
  },
  "errorsByFile": {}
}
```

Package.json Scripts

```
{
  "scripts": {
    "build": "tsbuild",
    "build:prod": "tsbuild --disallowimplicitany",
    "typecheck": "tsbuild check",
    "pretest": "tsbuild emitcheck 'test/**/*.ts'"
  }
}
```

Troubleshooting

Common Issues

"Cannot find module" errors in compiled output

Make sure path mappings are configured in tsconfig.json. tsbuild automatically transforms them.

Decorator errors

Ensure your tsconfig.json has:

```
{
  "compilerOptions": {
    "experimentalDecorators": true,
    "emitDecoratorMetadata": true
  }
}
```

Slow compilation

Use `--skiplibcheck` to skip declaration file checking:

```
npx tsbuild --skiplibcheck
```

Only use this if you trust your dependencies' type definitions.

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.

changelog.md for @git.zone/tsbuild

2026-03-24 - 4.4.0 - feat(config)

add smartconfig metadata and update TypeScript build configuration docs

- replace npmextra.json with .smartconfig.json and include it in published files
- add a repository license file
- upgrade TypeScript and related build dependencies
- refine tsconfig path handling to avoid mutating source path mappings
- fix tspublish config caching to distinguish unloaded and missing states
- expand documentation for import path rewriting, structured logging, and the compilation pipeline

2026-03-06 - 4.3.0 - feat(mod_logger)

add centralized TsBuildLogger and replace ad-hoc console output with structured, colored logging

- Add ts/mod_logger/classes.logger.ts providing header/step/detail/success/error/warn/indent utilities with ANSI color support
- Export logger from ts/mod_logger/index.ts and re-export from ts/index.ts
- Replace console.log/console.error/console.warn calls in mod_cli, mod_compiler, and mod_unpack with TsBuildLogger methods for consistent, hierarchical output
- Refactor error-summary, emit and type-check output to use logger separators, colors, and structured messages

2026-03-05 - 4.2.6 - fix(meta)

no changes

- Current package version: 4.2.5
- No code or file changes detected in this commit; no release required

2026-03-05 - 4.2.5 - fix(compiler)

yield to the event loop after TypeScript emit to allow pending microtasks and I/O to settle before reading or modifying the output directory

- Added await new Promise(resolve => process.nextTick(resolve)) immediately after program.emit()
- Prevents race conditions by allowing libuv write completions and other deferred callbacks to complete before accessing the output directory
- File changed: ts/mod_compiler/classes.tscompiler.ts

2026-03-05 - 4.2.4 - fix(fshelpers)

remove outdated comment about using synchronous rm to avoid XFS metadata corruption

- Comment-only change in ts/mod_fs/classes.fshelpers.ts; no runtime or API behavior changes
- Bump patch version from 4.2.3 to 4.2.4

2026-03-05 - 4.2.3 - fix(compiler)

defer unpacking until after all compilations and remove diagnostic filesystem syncs to avoid XFS metadata visibility issues

- Queue pending unpack operations during compilation and run them after all compile tasks complete to avoid modifying output directories while other compilations are writing.
- Remove TypeScript sys interception, execSync('sync') calls, and per-unpack fs.fsyncSync usage that attempted to work around XFS delayed metadata commits; rely on performing all unpacks after compilation instead.
- Clean up noisy diagnostic code (external 'ls' comparisons, readdir snapshots) and simplify logging of unpack results.
- Remove unused imports (fs and child_process.execSync) from the compiler module.

2026-03-05 - 4.2.2 - fix(compiler)

force global filesystem sync to flush XFS delayed logging and add diagnostics comparing Node's readdirSync with system ls to detect directory entry inconsistencies

- Replace per-directory fs.fsyncSync loop with execSync('sync') to ensure parent B+tree metadata is flushed on XFS
- Import execSync from child_process
- Add diagnostic comparison: run ls -1 and compare its entries to fs.readdirSync; log mismatches and full entry lists for debugging Node.js caching/readdir inconsistencies

2026-03-05 - 4.2.1 - fix(compiler)

use TypeScript sys hooks instead of fs monkeypatching to detect writes/deletes in previous output directories

- Replace direct fs.* monkeypatching with interception of typescript.sys.writeFile, typescript.sys.deleteFile and typescript.sys.createDirectory
- Add guards for optional sys.deleteFile before overriding it and preserve original sys methods to restore after compilation
- Update diagnostic messages to reference TypeScript sys ops and add an informational message when no ops are observed
- Reduce surface area of changes by avoiding global fs changes and focusing on TypeScript's sys API for safer interception

2026-03-05 - 4.2.0 - feat(mod_compiler)

add diagnostic interception of fs operations to detect and report unexpected file system changes in previously compiled output directories during compilation

- Wraps fs.unlinkSync, fs.rmSync, fs.rmdirSync, fs.renameSync and fs.writeFileSync to record operations targeting other successful output directories during a compile.
- Enabled only when there are previously compiled output dirs and when not running in quiet or JSON mode; original fs methods are restored after compilation.
- Logs up to 30 intercepted operations and prints a summary count if any ops were observed; intercepted calls still perform the original fs action (diagnostic-only).
- No functional change to compilation behavior beyond additional diagnostic reporting.

2026-03-05 - 4.1.26 - fix(compiler)

fsync output directories after unpack to avoid XFS delayed logging causing corrupt or invisible directory entries during subsequent TypeScript emits

- Force fsync on each successful output directory (open, fsync, close) before the next compilation step.
- Prevents XFS delayed logging from making directory entries invisible or corrupted during TypeScript emit operations.
- Operation swallows errors if a directory doesn't exist yet; non-breaking fix with small additional IO.

2026-03-05 - 4.1.25 - fix(mod_unpack)

flush directory metadata on XFS before reading and use readdirSync-based iteration to avoid missing entries when unpacking

- Call fs.fsyncSync on destination and nested directory file descriptors to force XFS to commit delayed directory metadata (addresses XFS CIL delayed logging causing incomplete readdir/opendir results).
- Replace opendirSync/readSync loops with readdirSync-based iteration for simpler, deterministic directory listing.
- Remove unused moved counter and update diagnostic log to report nestedEntries.length for moved entry count.

2026-03-05 - 4.1.24 - fix(mod_unpack)

iterate directories with opendirSync/readSync to avoid missing entries on XFS and ensure directory handles are closed

- Replaced readdirSync loops with opendirSync + readSync for destination and nested directories to provide a single stable directory handle during iteration
- Added explicit closeSync() calls to close directory handles and avoid resource leaks

- Avoids partial results/missed entries that can occur when repeatedly calling readdirSync (observed on XFS with delayed metadata)
- Preserves existing renameSync move logic and increments moved counter while cleaning up the now-empty nested directory

2026-03-05 - 4.1.23 - fix(mod_unpack)

handle partial readdirSync results when moving nested directory entries and add diagnostic log

- Loop over readdirSync results until the nested directory is empty to avoid missing entries from partial reads
- Count moved entries and print a diagnostic message with the final destination entry count
- Keep removal of the now-empty nested directory (fs.rmdirSync) after moving contents

2026-03-05 - 4.1.22 - fix(mod_compiler)

improve logging of successful output directories to include a sorted list of entries and use a shortened relative path

- Adds shortDir variable to display relative path instead of repeating inline replace(this.cwd + '/')
- Appends a sorted, comma-separated list of directory entries to the log output for easier inspection
- Change located in ts/mod_compiler/classes.tscompiler.ts

2026-03-05 - 4.1.21 - fix(compiler)

log emitted files written outside expected destination directory for diagnostics

- Adds diagnostic logging for emitted files that are not under the configured destDir, listing up to 20 example paths and reporting the remaining count.
- Logging is conditional: only when not in quiet mode and not emitting JSON.

- Diagnostic runs after compilation (post-compile) and before unpacking of outputs; paths are trimmed using the process cwd for readability.

2026-03-05 - 4.1.20 - fix(mod_compiler)

add diagnostic snapshots for output directories around clear and compile steps

- Introduce diagSnap helper to log entry and directory counts for successful output directories when not in quiet or JSON mode
- Call diagSnap before clearing the destination directory, after clearing, and after compilation to aid debugging of missing or unexpected outputs
- Behavior for emitted files and unpacking is unchanged; this is observational/logging-only instrumentation

2026-03-05 - 4.1.19 - fix(mod_fs)

use synchronous rm to avoid XFS metadata corruption when removing directories

- Replaced async fs.promises.rm with synchronous fs.rmSync in removeDirectory to avoid observed XFS metadata corruption affecting sibling entries under libuv thread-pool and signal pressure
- Retains previous options: recursive, force, maxRetries, retryDelay
- Adds inline comment documenting the rationale for using a synchronous removal

2026-03-05 - 4.1.18 - fix(mod_compiler)

add diagnostic logging of output directory states after compilation and after import-path rewriting to aid debugging

- Imported fs to allow reading output directories for diagnostics
- Logs entries and directory counts for each successful output directory both pre- and post-import-path-rewrite
- Diagnostics are gated by !isQuiet && !isJson and are read-only (no behavior change)

- Tags used: 'diag' (post-compilation) and 'diag-post-rewrite' (after rewriting) to help identify missing or unexpected output folders

2026-03-05 - 4.1.17 - fix(tsunpacker)

use synchronous fs operations in tsunpacker to avoid readdir race conditions

- Replaced async fs.promises.readdir/rename/rm/rmdir loops with fs.readdirSync/renameSync/rmSync/rmdirSync
- Removed readdir retry loops that attempted to handle partial/stale readdir results
- Updated comment to document rationale: avoid race conditions under signal pressure and XFS metadata lag
- Note: function remains async but now performs blocking sync filesystem calls which may block the event loop during unpack

2026-03-05 - 4.1.16 - fix(mod_unpack)

handle partial readdir results from signal-interrupted getdents64 when unpacking to ensure sibling removal and nested moves complete

- Loop readdir calls for destination directory until only the source folder remains to avoid partial-listing leftovers
- Loop readdir calls for nested directory and repeatedly rename entries until the nested directory is empty
- Prevents leftover files and incomplete moves when readdir returns partial results under signals

2026-03-05 - 4.1.15 - fix(mod_unpack)

flatten nested output directory without temporary rename steps to avoid race conditions

- Replace rename-rm-rename strategy with: remove sibling entries in destination, move nested source entries up into the destination, then remove the now-empty nested folder.
- Avoid creating temporary sibling directories and avoid removing the destination directory to reduce filesystem race conditions and metadata lag issues (XFS/NFS/etc.).
- Remove removed removeEmptyDirectory helper and stop using FsHelpers.move/removeDirectory in unpack; import and use fs.promises methods (readdir, rm, rename, rmdir) directly.

2026-03-05 - 4.1.14 - fix(fs)

replace execSync and fsync workarounds with atomic async FsHelpers operations to avoid XFS races and shell dependencies

- Removed child_process.execSync usage and shell mv/rm commands in mod_unpack and mod_compiler.
- Removed syncDirectoryTree and fsync-based workaround from the compiler module.
- Use FsHelpers.move and FsHelpers.removeDirectory (async rename/remove) for atomic filesystem operations during unpack.
- Await performUnpack directly and simplify unpack flow to improve portability and reliability on XFS and other filesystems.

2026-03-05 - 4.1.13 - fix(mod_unpack)

Use child_process.execSync (mv/rm) to perform unpack atomically, replacing async fs operations and logs to avoid ENOENT/partial directory listings on XFS

- Replaced async fs.promises.rename/rm and readdir/stat debugging with execSync rm -rf and mv operations for sequential, atomic moves
- Imported execSync from child_process and removed verbose console logging and extra fs checks
- Addresses race conditions observed on filesystems like XFS where libuv async operations can return partial results or ENOENT errors

2026-03-05 - 4.1.12 - fix(mod_compiler)

replace runtime require calls with top-level imports and use execSync/path.join for filesystem sync and traversal

- Added top-level imports: path and execSync from child_process
- Replaced require('child_process').execSync('sync') with execSync('sync') to force fs sync
- Replaced require('path').join(...) with path.join(...) when recursing directories
- Refactor is purely local/maintenance-focused (consistency and slight performance/readability improvement); no functional change expected

2026-03-05 - 4.1.11 - fix(mod_compiler)

flush directory entries before unpack to avoid XFS delayed-log causing partial readdir results

- Add fs import and call child_process.execSync('sync') before unpacking compiled output to force kernel-level sync
- Add syncDirectoryTree(dir) to recursively open and fsync directory file descriptors and traverse subdirectories
- Call syncDirectoryTree on the destination directory before performing unpack to ensure TypeScript writeFileSync entries are committed (prevents partial readdir results on XFS)
- Errors during directory sync are ignored to avoid breaking normal flow

2026-03-05 - 4.1.10 - fix(unpack)

use atomic renames to flatten nested output and make unpacking more reliable

- Replace per-entry moves with an atomic 3-step strategy: rename nested dir to a temp location, remove the destination dir, then rename temp back to the destination to avoid partial readdir/move under signal pressure.
- FsHelpers.move switched from sync rename to fs.promises.rename to work with async flows.
- Use fs.promises.rm with retries and explicit temp-dir cleanup to handle previous failed runs.
- Add diagnostic logging and verification of intermediate states.
- Removed the older removeSiblingDirectories and moveNestedContentsUp methods in favor of the new rename-based approach.

2026-03-05 - 4.1.9 - fix(fs)

improve filesystem helpers: use sync rename for reliability on certain filesystems; retry rmdir with delays and avoid recursive rm; bump @push.rocks/smartfs to ^1.3.2

- move(): use fs.renameSync to improve reliability on XFS/mounted filesystems where async rename can be interrupted by process managers
- removeEmptyDirectory(): retry fs.promises.rmdir up to 5 attempts with incremental delays; return on ENOENT; avoid recursive rm on ENOTEMPTY to prevent removing still-valid references
- Dependency bump: @push.rocks/smartfs from ^1.3.1 to ^1.3.2

2026-03-05 - 4.1.8 - fix(unpack)

catch unpack errors and add verbose unpack logging

- Wrap performUnpack call in the compiler to catch and log exceptions so unpack failures don't crash the process
- Log the number of directories and files being unpacked when moving nested contents to aid debugging

2026-03-05 - 4.1.7 - fix(fs/compiler/unpack)

robustify directory removal and remove noisy diagnostic logging

- Use fs.promises.rm with force, maxRetries and retryDelay in removeDirectory to reduce intermittent failures during removals.
- Handle ENOTEMPTY in removeEmptyDirectory by falling back to a recursive rm and ignore ENOENT to avoid errors from transient filesystem metadata lag.
- Remove several diagnostic/verbose console logs in the compiler and unpacker paths to reduce noisy output and simplify flow.
- Short-circuit unpack logic when no nesting is detected to avoid unnecessary work and logs.
- No public API or behavior-breaking changes; suitable for a patch release.

2026-03-05 - 4.1.6 - fix(mod_compiler)

add diagnostic logging to report dist_ts and output directory contents after each compilation task and after import-path rewriting

- Adds DIAG-CROSSTASK logs to inspect dist_ts subdirectories after each task when not in quiet or JSON mode
- Adds DIAG-FINAL logs to report directory names and file counts for each successful output dir after import-path rewriting
- Diagnostics use dynamic fs import and are non-intrusive: gated by isQuiet/isJson and errors are ignored

2026-03-05 - 4.1.5 - fix(diagnostics)

add diagnostic logging around compilation and unpack to aid troubleshooting

- Enable TypeScript CompilerOptions.listEmittedFiles to surface emitted file info
- Log destination directory contents and emitted file/error counts after compile and after unpack (only when not quiet and not json)
- Add unpack-specific diagnostic logs: shouldUnpack skip, detectNesting result and nestedPath, and nested/destination directory listings before removing sibling directories
- All logs use console.log and are wrapped in try/catch to avoid throwing; changes are informational and non-breaking

2026-03-05 - 4.1.4 - fix(deps)

bump @git.zone/tspublish dependency to ^1.11.2

- Updated @git.zone/tspublish from ^1.11.0 to ^1.11.2 in package.json

2026-03-04 - 4.1.3 - fix(deps)

bump dependencies: @push.rocks/smartcli, @push.rocks/smartlog, @git.zone/tstest, and @types/node to their newer versions

- @push.rocks/smartcli: ^4.0.19 -> ^4.0.20
- @push.rocks/smartlog: ^3.1.10 -> ^3.2.1
- @git.zone/tstest: ^3.1.4 -> ^3.2.0
- @types/node: ^25.0.3 -> ^25.3.3
- Non-breaking dependency version bumps

2026-01-04 - 4.1.0 - feat(docs)

update README with improved docs and monorepo/tspublish guidance; namespace and extend npmextra.json with release registries; bump several dependencies

- Bumped dependencies: @git.zone/tspublish ^1.10.3 → ^1.11.0, @push.rocks/smartfs ^1.2.0 → ^1.3.1, @git.zone/tstest ^3.1.3 → ^3.1.4, @types/node ^25.0.1 → ^25.0.3
- npmextra.json reorganized: replaced legacy keys with namespaced entries (@git.zone/cli, @git.zone/tsdoc) and added @ship.zone/szci; added release configuration with registries [https://verdaccio.lossless.digital, https://registry.npmjs.org] and public accessLevel
- README rewritten: improved formatting, added emojis and tables, new 'Issue Reporting and Security' section, monorepo/tspublish usage docs, auto-unpack and decorator guidance, CI/CD and troubleshooting improvements
- Non-breaking changes; recommended next semantic version bump is minor

2025-12-14 - 4.0.2 - fix(TsCompiler)

Clear output directories before compilation to ensure clean builds and avoid stale files

- TsCompiler.compileGlob now clears the destination directory (if it exists) before compiling each glob pattern.
- Clearing is logged unless --quiet or --json flags are set (e.g. "□□ Clearing output directory: ").
- Uses FsHelpers.removeDirectory to remove previous output, preventing stale or duplicate emitted files.
- Documentation (readme.md) updated to advertise automatic output directory management / clean builds.
- Removed stale compiled test artifacts from test/assets/output to avoid interference with tests.

2025-12-13 - 3.1.3 - fix(npmextra)

Align npmextra.json package name with package.json (@git.zone/tsbuild)

- Corrected npmPackagename in npmextra.json from "@gitzone/tsbuild" to "@git.zone/tsbuild" to match package.json and README
- Metadata-only change: no code or API behavior affected

2025-11-28 - 3.1.2 - fix(TsBuild)

Set default TypeScript target to ESNext

- Default compiler target changed from ScriptTarget.ES2024 to ScriptTarget.ESNext in ts/tsbuild.classes.tsbuild.ts
- Aligns the default compiler options with documentation and ensures use of the latest language features

2025-11-27 - 3.1.1 - fix(compiler)

Update default TypeScript target to ES2024

- Default compiler option 'target' changed from ScriptTarget.ESNext to ScriptTarget.ES2024 in ts/tsbuild.classes.tsbuild.ts
- Aligns emitted code to ES2024 language features by default

2025-11-17 - 3.1.0 - feat(tsbuild.classes)

Update default TypeScript lib to lib.esnext.d.ts

- Changed default compilerOptions.lib from ['lib.dom.d.ts', 'lib.es2022.d.ts'] to ['lib.dom.d.ts', 'lib.esnext.d.ts'] in compilerOptionsDefault.
- Allows newer ECMAScript/DOM features by default when compiling with tsbuild (affects emitted types and available globals).

- Behavioral default change only — no public API changes; callers can still override lib via tsconfig, programmatic options, or CLI.

2025-11-17 - 3.0.0 - BREAKING CHANGE(TsBuild)

Stop forcing emitDecoratorMetadata in protected compiler defaults

- Removed emitDecoratorMetadata from the set of protected/critical default compiler options.
- Projects that relied on tsbuild automatically enabling emitDecoratorMetadata (for DI frameworks like NestJS, TypeORM, Inversify) must now enable it explicitly in their tsconfig.json or via programmatic/CLI options.
- No other protected defaults were changed; outDir, noEmitOnError, declaration and inlineSourceMap remain enforced.
- Behavior is now more permissive: decorator metadata generation is controlled by user configuration rather than being forced by tsbuild.

2025-11-17 - 2.7.3 - fix(tsbuild.classes)

Remove duplicate emitDecoratorMetadata from default compiler options and centralize it in protected defaults

- Removed emitDecoratorMetadata from compilerOptionsDefault in ts/tsbuild.classes.tsbuild.ts to avoid duplicate configuration.
- emitDecoratorMetadata remains enforced via getCriticalDefaults(), ensuring decorator metadata support is protected from tsconfig.json overrides.
- Prevents inconsistencies during compiler option merging by centralizing the decorator-related setting.

2025-11-17 - 2.7.2 - fix(compilerOptions)

Remove experimentalDecorators and useDefineForClassFields from default TypeScript compiler options

- Removed experimentalDecorators from compilerOptionsDefault in ts/tsbuild.classes.tsbuild.ts
- Removed useDefineForClassFields from compilerOptionsDefault in ts/tsbuild.classes.tsbuild.ts
- Default compiler options now rely on TypeScript's upstream defaults for decorator and class field behavior
- If your project relies on these settings, re-enable them in your tsconfig.json or pass them via the programmatic API / CLI

2025-11-02 - 2.7.1 - fix(readme)

Update documentation: expand README with usage, CLI and API examples; add readme.hints.md project memory

- Add readme.hints.md: new project memory / quick reference with public API and CLI summaries
- Expand and restructure readme.md: more comprehensive Quick Start, CLI Commands, API Reference, configuration, examples and troubleshooting
- Clarify protected compiler options, default compiler options, path transformation behavior and error-handling patterns
- Docs-only change — no source code or behavioral changes

2025-11-02 - 2.7.0 - feat(tsbuild)

Add tsconfig.json support and safer compiler option merging; protect critical options; apply path and enum transforms; bump dependencies.

- Add robust tsconfig.json reading with graceful fallback when no tsconfig is present or it is invalid
- Merge compiler options in a clear priority order (defaults -> tsconfig -> protected defaults -> programmatic -> CLI flags)
- Introduce protected (critical) compiler options that cannot be overridden by tsconfig.json: outDir, noEmitOnError, declaration, emitDecoratorMetadata, inlineSourceMap
- Convert string values from tsconfig (target, module, moduleResolution) to TypeScript enum values where applicable; special-case NodeNext
- Transform tsconfig path mappings by replacing './ts_' with './dist_ts_' to keep runtime path resolution consistent with compiled output

- Expose getCriticalDefaults helper and adjust mergeCompilerOptions to apply protected defaults before programmatic and CLI overrides
- Update README with documentation for tsconfig support, merge order, protected compiler options, and example tsconfig
- Bump dependencies/devDependencies: @push.rocks/smartcli ^4.0.19, @push.rocks/smartlog ^3.1.10, typescript 5.9.3, @git.zone/tsrun ^1.6.2, @git.zone/tstest ^2.7.0

2025-08-29 - 2.6.8 - fix(tsbuild)

Avoid process.exit in library, add confirmskiplibcheck flag, improve CLI exit handling and JSON/quiet modes, update test script

- Changed package.json test script from "tsrun test/test.ts --verbose" to "tstest test/test.ts --verbose".
- Library no longer calls process.exit from compile and compileWithErrorTracking; errors are returned or thrown so callers can decide process termination.
- skipLibCheck behavior updated: delay/warning only happens when --confirmskiplibcheck is present; otherwise a short informational note is printed (suppressed in --quiet/--json).
- CLI now awaits compileGlobStringObject calls and inspects a final error summary attached to argv to decide process.exit(1) when errors occurred.
- compileGlobStringObject/exports now respect --quiet and --json modes, emit a JSON summary when --json is used, and attach the final error summary to argv so the CLI can determine exit behavior.

2025-08-18 - 2.6.7 - fix(tspublish)

Bump @git.zone/tspublish dependency to ^1.10.3

- Updated dependency @git.zone/tspublish from ^1.10.2 to ^1.10.3 in package.json

2025-08-18 - 2.6.6 - fix(dependencies)

Update dependency @git.zone/tspublish to ^1.10.2

- Bumped @git.zone/tspublish in package.json from ^1.10.1 to ^1.10.2

2025-08-18 - 2.6.5 - fix(dependencies)

Bump dependencies and add pnpm-workspace configuration

- Updated @git.zone/tspublish from ^1.9.1 to ^1.10.1
- Updated @push.rocks/smartfile from ^11.2.4 to ^11.2.7
- Updated @push.rocks/smartpath from ^5.0.18 to ^6.0.0
- Updated typescript from 5.8.3 to 5.9.2
- Updated devDependency @git.zone/tstest from ^1.10.1 to ^2.3.4
- Added pnpm-workspace.yaml with onlyBuiltDependencies list (esbuild, mongodb-memory-server, puppeteer)

2025-05-24 - 2.6.4 - fix(dependencies)

Add .npmrc and update dependency versions for smartfile and tstest

- Add .npmrc with registry configuration for npm
- Bump @push.rocks/smartfile version from ^11.2.3 to ^11.2.4
- Bump @git.zone/tstest version from ^1.9.0 to ^1.10.1

2025-05-21 - 2.6.3 - fix(tsbuild)

minor maintenance updates and documentation improvements

- Updated commit metadata to align with project version
- Refined CLI command parsing and diagnostics logging for better clarity
- Improved code readability in compiler options merging

2025-05-21 - 2.6.2 - fix(npm configuration)

Remove .npmrc file to default npm registry behavior

- Deleted .npmrc file that hard-coded the npm registry URL to https://registry.npmjs.org/
- This change leverages npm's default registry settings and reduces configuration clutter

2025-05-21 - 2.6.1 - fix(tsbuild.classes)

Improve error diagnostics handling by removing legacy helper and integrating more robust error summaries in the compilation process

- Removed the handleDiagnostics method and its legacy usage
- Replaced legacy calls with inline processDiagnostics checks for pre-emit and emit phases
- Combined error summaries for clearer reporting during emit checks and type validation
- Enhanced error output to guide users on resolving TypeScript errors before emission

2025-05-21 - 2.6.0 - feat(tsbuild)

Improve task logging and update dependencies

- Add .npmrc file with npm registry configuration
- Update test script to use '--verbose' flag
- Bump dependency versions for @push.rocks packages and TypeScript
- Enhance TsBuild logging by incorporating task info (e.g. task number, total tasks, output folder, and compile durations)
- Propagate task info in compileFileArrayWithErrorTracking for better task tracking

2025-05-21 - 2.5.2 - fix(tsbuild)

Improve diagnostic error handling and summary reporting for TypeScript compilation by refactoring diagnostic processing and adding pre-emit error checks.

- Introduce a dedicated processDiagnostics function that categorizes errors by file and computes error totals.
- Refactor displayErrorSummary to provide clearer, color-coded output of error details.
- Add pre-emit error checking in compileWithErrorTracking to prevent emission when errors exist.

- Consolidate error summary merging in compileFileArrayWithErrorTracking for improved reporting.

2025-05-15 - 2.5.1 - fix(commitinfo)

Update commit information and metadata to synchronize release data

- Regenerated the commitinfo file with current version details
- Maintained existing functionality with no functional code changes

2025-05-15 - 2.5.0 - feat(cli)

Enhance type checking in CLI by adding default file pattern handling

- When no TypeScript file or glob pattern is provided, the CLI now performs a default type checking sequence.
- First checks 'ts/**/*' **files with standard options, then checks 'test/**/*' files with skiplibcheck enabled.
- Improved logging to indicate file discovery and check results, ensuring clear feedback for users.

2025-05-15 - 2.4.1 - fix(cli)

Improve TS folder compilation order display in CLI

- Refactor folder compilation output to use a bordered, tabular format with order numbering
- Enhance readability of TS folder compilation plan in the CLI output

2025-05-15 - 2.4.0 - feat(cli)

Add new 'check' command for type checking and update compiler options handling

- Introduced a new 'check' command to verify TypeScript files without emitting output
- Updated CLI error messages and logging for better clarity

- Replaced '--allowimplicitany' flag with '--disallowimplicitany' to reflect new default behavior
- Modified compiler options default settings (noImplicitAny now set to false) for more flexible type handling
- Refined diagnostic output in tsbuild class for improved error reporting
- Updated .gitignore to exclude the .claude file
- Enhanced documentation in readme and implementation plan files

2025-03-20 - 2.3.2 - fix(compileGlobStringObject)

Fix duplicate file outputs in glob pattern processing

- Removed duplicate concatenation of compiled files in compileGlobStringObject
- Ensured unique file paths are used during TypeScript compilation

2025-03-20 - 2.3.1 - fix(compiler)

Refactor compiler implementation with consolidated TsBuild class and improved diagnostics handling

- Removed legacy tsbuild.classes.compiler.ts and introduced tsbuild.classes.tsbuild.ts
- Unified compiler options merging, reading tsconfig.json, and diagnostics reporting within the TsBuild class
- Updated exports to reference the new compiler class implementation for backward compatibility

2025-03-20 - 2.3.0 - feat(cli)

Add emitcheck command to validate TS file emission without generating output

- Implemented the emitcheck CLI command to allow checking if TypeScript files can be emitted without producing files
- Updated tsbuild.classes.compiler.ts to include the emitCheck function
- Enhanced documentation in readme.md to describe the new emitcheck usage with examples

2025-03-17 - 2.2.7 - fix(compiler)

Improve diagnostic checking and error handling in the TypeScript compiler integration

- Added pre-emit diagnostics check to log errors before emitting
- Process exits on encountering pre-emit errors to ensure build correctness
- Enhanced logging for emit diagnostics to aid debugging

2025-03-04 - 2.2.6 - fix(package)

Fix repository URL in package.json

- Updated repository URL in package.json to point to the correct Git repository.

2025-03-04 - 2.2.5 - fix(package.json)

Update repository URLs in package metadata.

- Corrected the 'bugs.url' and 'homepage' fields with the accurate URLs.

2025-03-04 - 2.2.4 - fix(core)

Fix compiler logic to remove duplicate compiled files and improve glob pattern handling.

- Resolved an issue causing duplicate file compilations when using compileGlobStringObject.
- Improved handling of glob patterns to ensure accurate compilation paths.

2025-03-04 - 2.2.3 - fix(exports)

Fixed duplicate file compilation in compileGlobStringObject

- Enhanced type safety checks for glob pattern compilation keys.
- Ensured file list transformations include type checks.
- Fixed issue with duplicate compilation results in compileGlobStringObject.

2025-01-28 - 2.2.2 - fix(ci)

Remove GitLab CI configuration

2025-01-28 - 2.2.1 - fix(core)

Update dependencies to improve stability and performance.

- Updated @git.zone/tspublish from version ^1.7.5 to ^1.9.1.
- Updated @push.rocks/smartfile from version ^11.0.21 to ^11.1.5.
- Updated @push.rocks/smartpromise from version ^4.0.4 to ^4.2.2.
- Updated typescript from version 5.6.3 to 5.7.3.
- Updated @push.rocks/tapbundle from version ^5.0.23 to ^5.5.6.
- Updated @types/node from version ^22.8.7 to ^22.12.0.

2024-11-05 - 2.2.0 - feat(cli)

Enhance CLI for TypeScript folder compilation ordering based on rank and predefined rules.

- CLI now supports automatic ordering of TypeScript folders for compilation using tspublish.json based ranking.
- Ensures 'ts_interfaces' and 'ts_shared' are always transpiled first if certain conditions are met.
- Updated TypeScript compilerOptions to support additional path transformations.
- Updated dependencies versions and added '@git.zone/tspublish' in ts/plugins.ts.

2024-10-27 - 2.1.85 - fix(compiler)

Improve path handling in compiler options

- Refactored path import in tsbuild.classes.compiler.ts.
- Enhanced mergeCompilerOptions to read paths and baseUrl from tsconfig.json if present.

2024-07-22 - 2.1.84 - fix(cli)

Fixed transpilation order issue in tsfolders command

- Corrected the transpilation order so 'ts_shared' is processed before other folders in the 'tsfolders' CLI command.

2024-07-21 - 2.1.83 - fix(cli)

Ensure 'ts_shared' folder is compiled first if present

- Added logic to make sure the 'ts_shared' folder is compiled first when running 'tsfolders' command.

2024-06-24 - 2.1.82 - fix(core)

Minor improvements and optimizations in core TypeScript compiler integration.

- Updated TypeScript dependency to latest version 5.5.2.
- Enhanced logging in compiler process.
- Improved handling of CLI commands for better flexibility.
- Compiled output directories are now more structured.
- Refactored internal compiler options merge function.

2024-06-24 - 2.1.81 - fix(dependencies)

Update dependencies to latest versions

- Upgraded @push.rocks/smartcli from ^4.0.10 to ^4.0.11
- Upgraded @push.rocks/smartfile from ^11.0.14 to ^11.0.21
- Upgraded @push.rocks/smartlog from ^3.0.3 to ^3.0.7
- Upgraded @push.rocks/smartpromise from ^4.0.3 to ^4.0.4
- Upgraded typescript from 5.4.5 to 5.5.2
- Upgraded @git.zone/tsrun from ^1.2.46 to ^1.2.47

- Upgraded @types/node from ^20.12.11 to ^20.14.8

2024-05-17 - 2.1.80 - core

Fix multiple core issues.

- Various small fixes and updates to the core functionality.

2024-05-17 - 2.1.76 to 2.1.79 - core

Routine core updates and fixes.

- Several minor enhancements and bug fixes.

2024-05-14 - 2.1.74 to 2.1.75 - core

General core maintenance updates.

- Core updated to fix minor bugs.

2024-05-10 - 2.1.72 to 2.1.73 - core

Minor core updates.

- Improvements and fixes to core components.

2024-01-08 - 2.1.70 to 2.1.71 - core

Core functionality enhancements.

- Small fixes and updates in the core.

2023-08-26 - 2.1.66 to 2.1.69 - core

Regular core updates and fixes.

- Various updates to improve core functionality.

2023-06-03 - 2.1.65 - core

Core maintenance update.

- Fixed issues in core functionality.

2022-08-03 - 2.1.63 to 2.1.64 - core

Minor core updates and fixes.

- Updated core functionality with minor fixes.

2022-05-25 - 2.1.61 to 2.1.62 - core

Routine core updates.

- Fixed minor bugs in the core.

2022-03-24 - 2.1.60 - core

Core functionality update.

- Fixed various minor issues in core.

2022-03-18 - 2.1.57 to 2.1.59 - core

Minor core updates and enhancements.

- Updated core components with small fixes.

2022-03-15 - 2.1.50 to 2.1.56 - core

Several core bug fixes.

- Fixed various minor issues in the core functionality.

2022-03-14 - 2.1.49 - core

Core update.

- Fixed minor bugs in the core.

2022-03-12 - 2.1.43 to 2.1.48 - core

General core maintenance updates.

- Core updated to fix minor bugs and improve functionality.

2022-03-11 - 2.1.33 to 2.1.42 - core

Core functionality enhancements and fixes.

- Multiple updates to improve core functionality and fix bugs.

2022-03-11 - 2.1.29 to 2.1.32 - core

Routine core updates.

- Minor bug fixes and improvements in core functionality.

2022-01-19 - 2.1.28 to 2.1.29 - core

Core bug fixes.

- Updated core to address minor issues.

2021-10-06 - 2.1.27 - core

Minor core update.

- Fixed core bugs.

2021-09-08 - 2.1.26 - core

Core update.

- Fixed minor bugs in the core.

2021-08-17 - 2.1.25 - core

Core bug fixes.

- Small updates to improve core functionality.

2020-08-11 - 2.1.24 - core

Routine core update.

- Fixed minor bugs in the core.

2020-05-14 - 2.1.23 - core

General core updates.

- Fixes to improve core stability.

2020-03-13 - 2.1.20 to 2.1.22 - core

Minor core updates.

- Enhancements and fixes for core functionality.

2020-03-09 - 2.1.19 - core

Maintenance core update.

- Fixes addressing minor issues in core functionality.

2019-08-26 - 2.1.16 to 2.1.17 - core

Routine minor core updates.

- Improvement and fixes within the core functionality.

2019-01-27 - 2.1.6 - custom directory compilation

Now picking up TypeScript files correctly.

- Resolved compilation issues with custom directories.

2018-12-05 - 2.1.0 to 2.1.1 - core

Minor core updates.

- Small enhancements and fixes applied to core functionality.

2018-12-05 - 2.0.22 - cli options

Now support --web for web compilations targeting Google Chrome.

- Added new CLI option for web compilation.

2018-07-25 - 2.0.15 to 2.0.19 - various

Multiple fixes across core, dependency, and compiler modules.

- Packagename fix in core.
- Dependency updates.
- Compiler options fixed.
- Initial core updates.