

readme.md for @push.rocks/smartconfig

A comprehensive TypeScript configuration management library providing centralized tool configs, persistent key-value storage, and powerful environment variable mapping with automatic type conversions.

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

```
npm install @push.rocks/smartconfig --save
# or
pnpm add @push.rocks/smartconfig
```

Quick Start

```
import { Smartconfig, AppData, KeyValueStore } from '@push.rocks/smartconfig';

// 1. Read tool config from .smartconfig.json
const sc = new Smartconfig();
const eslintOpts = sc.dataFor('eslint', { extends: 'standard' });

// 2. Map env vars to typed config (with auto-conversion)
```

```
const appData = await AppData.createAndInit<{ port: number; debug: boolean }>({
  envMapping: {
    port: 'PORT',
    debug: 'boolean:DEBUG',
  },
});

// 3. Persist data between runs
const kv = new KeyValueStore({ typeArg: 'userHomeDir', identityArg: 'myapp' });
await kv.writeKey('lastRun', Date.now());
```

Three Core Classes

1. `Smartconfig` — Centralized Tool Configuration

Reads a `.smartconfig.json` file from a project directory and merges its contents with your defaults. One file, every tool.

```
import { Smartconfig } from '@push.rocks/smartconfig';

const sc = new Smartconfig(); // uses cwd
const sc2 = new Smartconfig('/my/project'); // or specify a path

const prettierConfig = sc.dataFor<PrettierConfig>('prettier', {
  semi: false,
  singleQuote: true,
});
```

`.smartconfig.json` example:

```
{
  "prettier": {
    "semi": true,
    "printWidth": 120
  },
  "eslint": {
```

```
    "extends": "@company/eslint-config"
  }
}
```

Values from the file override the defaults you pass in. Missing keys fall back to your defaults.

Properties:

- `smartconfigJsonExists: boolean` — whether `.smartconfig.json` was found
- `smartconfigJsonData: any` — the parsed JSON contents

Methods:

- `dataFor<T>(toolName: string, defaults: T): T` — returns merged config

2. `KeyValueStore<T>` — Persistent Data Storage

A generic, typed key-value store that persists JSON to disk (or stays in-memory for tests). Supports change detection via RxJS observables.

```
import { KeyValueStore } from '@push.rocks/smartconfig';

interface Settings {
  username: string;
  theme: 'light' | 'dark';
}

// Store in ~/.smartconfig/kv/
const kv = new KeyValueStore<Settings>({
  typeArg: 'userHomeDir',
  identityArg: 'myApp',
  mandatoryKeys: ['username'],
});

await kv.writeKey('username', 'jane');
await kv.writeKey('theme', 'dark');

const user = await kv.readKey('username'); // 'jane'
```

```
const all = await kv.readAll(); // { username: 'jane', theme: 'dark' }

// React to changes
kv.changeSubject.subscribe((data) => console.log('changed:', data));
```

Storage types:

typeArg	Where it goes	Use case
'userHomeDir'	~/.smartconfig/kv/<identity>.json	CLI tools, per-user state
'custom'	Your path (file or directory)	App data, project-local state
'ephemeral'	Memory only — nothing on disk	Tests

Methods:

Method	Description
readKey(key)	Read a single value
writeKey(key, value)	Write a single value
readAll()	Read everything
writeAll(obj)	Merge an object into the store
deleteKey(key)	Remove a key
reset()	Wipe all keys (syncd to disk)
wipe()	Delete the backing file entirely
getMissingMandatoryKeys()	Returns keys declared mandatory but not yet set
waitForKeysPresent(keys)	Returns a Promise that resolves once all listed keys exist
waitForAndGetKey(key)	Waits for a key, then returns its value

3. AppData<T> — Environment Variable Mapping

The flagship class. Maps environment variables (or hardcoded values) into a typed config object with automatic type conversions, nested object support, and smart storage path selection.

```
import { AppData } from '@push.rocks/smartconfig';

interface Config {
```

```

apiUrl: string;
apiKey: string;
features: {
  analytics: boolean;
  payment: boolean;
};
redis: {
  host: string;
  password: string;
};
}

const appData = await AppData.createAndInit<Config>({
  requiredKeys: ['apiKey'],
  envMapping: {
    apiUrl: 'API_URL', // plain env var
    apiKey: 'hard:dev-key-123', // hardcoded fallback
    features: {
      analytics: 'boolean:ENABLE_ANALYTICS', // converts "true"/"false" → boolean
      payment: 'hard_boolean:true', // hardcoded boolean
    },
    redis: {
      host: 'REDIS_HOST',
      password: 'base64:REDIS_PASSWORD_B64', // base64-decode at load time
    },
  },
  overwriteObject: {
    apiUrl: 'http://localhost:3000', // force override after env mapping
  },
});

const store = await appData.getKvStore();
const url = await store.readKey('apiUrl');

```

Mapping Prefixes

Prefix	What it does	Example mapping	Result
(none)	Raw env var as string	'MY_VAR'	process.env.MY_VAR
hard:	Hardcoded string	'hard:hello'	"hello"

Prefix	What it does	Example mapping	Result
<code>boolean:</code>	Env var → <code>true</code> / <code>false</code>	<code>'boolean:FLAG'</code>	<code>true</code> or <code>false</code>
<code>json:</code>	Env var → <code>JSON.parse()</code>	<code>'json:CONFIG'</code>	parsed object
<code>base64:</code>	Env var → base64 decode	<code>'base64:SECRET'</code>	decoded string
<code>hard_boolean:</code>	Hardcoded boolean	<code>'hard_boolean:false'</code>	<code>false</code>
<code>hard_json:</code>	Hardcoded JSON	<code>'hard_json:{"a":1}'</code>	<code>{ a: 1 }</code>
<code>hard_base64:</code>	Hardcoded base64	<code>'hard_base64:SGVsbG8='</code>	<code>"Hello"</code>

Suffix detection also works: a mapping ending in `_JSON` or `_BASE64` triggers the corresponding transform automatically.

Boolean Conversion Rules

The `boolean:` prefix (and `hard_boolean:`) recognizes:

- **true:** `"true"`, `"1"`, `"yes"`, `"y"`, `"on"` (case-insensitive)
- **false:** `"false"`, `"0"`, `"no"`, `"n"`, `"off"` (case-insensitive)

Nested Objects

Mapping values can be objects — they are resolved recursively:

```
envMapping: {
  database: {
    host: 'DB_HOST',
    port: 'hard:5432',
    credentials: {
      user: 'DB_USER',
      password: 'base64:DB_PASS_B64',
      ssl: 'boolean:DB_SSL',
    },
  },
}
```

Smart Storage Path

When no `dirPath` is specified, AppData auto-selects:

1. `/app/data` — if it exists (containers)
2. `/data` — if it exists (alternate container path)
3. `.nokit/appdata` — local dev fallback

Or pass `ephemeral: true` for zero disk I/O (great for tests).

Static Helpers

Quick one-shot env var reads without creating an AppData instance:

```
const isEnabled = await AppData.valueAsBoolean('FEATURE_FLAG');
const config     = await AppData.valueAsJson<MyType>('CONFIG_JSON');
const secret     = await AppData.valueAsBase64('ENCODED_SECRET');
const url        = await AppData.valueAsString('API_URL');
const port       = await AppData.valueAsNumber('PORT');
```

Instance Methods

Method	Description
<code>getKvStore()</code>	Returns the underlying <code>KeyValueStore<T></code>
<code>logMissingKeys()</code>	Logs and returns any required keys that are missing
<code>waitForAndGetKey(key)</code>	Blocks until a key is present, then returns it

Security

AppData automatically redacts sensitive values in its console logs. Keys matching patterns like `secret`, `token`, `password`, `api`, `auth`, `jwt`, etc. are truncated. JWT tokens (starting with `eyJ`) are also detected and shortened. Your actual stored values are never modified — only log output is redacted.

Real-World Example

```
import { Smartconfig, AppData, KeyValueStore } from '@push.rocks/smartconfig';

interface CliConfig {
  githubToken: string;
  model: 'gpt-3' | 'gpt-4';
  cache: { enabled: boolean; ttl: number };
}
```

```
// Tool-level config from .smartconfig.json
const sc = new Smartconfig();
const toolDefaults = sc.dataFor('mycli', { defaultModel: 'gpt-3' });

// Env-mapped runtime config
const appData = await AppData.createAndInit<CliConfig>({
  requiredKeys: ['githubToken'],
  envMapping: {
    githubToken: 'GITHUB_TOKEN',
    model: 'hard:gpt-4',
    cache: {
      enabled: 'boolean:ENABLE_CACHE',
      ttl: 'hard:3600',
    },
  },
});

// Persistent user-level cache
const cache = new KeyValueStore({
  typeArg: 'userHomeDir',
  identityArg: 'mycli-cache',
});

// Check mandatory keys
const missing = await appData.logMissingKeys();
if (missing.length > 0) {
  console.error('Missing config – set these env vars and retry.');
```

```
  process.exit(1);
}

const store = await appData.getKvStore();
const settings = await store.readAll();
console.log(`Model: ${settings.model}, Cache: ${settings.cache.enabled}`);
```

License and Legal Information

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

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

Trademarks

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

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

Company Information

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

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

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

Revision #3

Created 2026-03-28 11:09:56 UTC by foss.global Team

Updated 2026-03-28 12:16:33 UTC by foss.global Team