@push.rocks/smartconfig

Enhances npm with additional configuration and tool management capabilities, including a key-value store for project setups.

- readme.md for @push.rocks/smartconfig
- changelog.md for @push.rocks/smartconfig

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 (synced 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
`boolean:`	Env var → `true`/`false`	`'boolean:FLAG'`	`true` or `false`
`json:`	Env var → `JSON.parse()`	`'json:CONFIG'`	parsed object
`base64:`	Env var → base64 decode	`'base64:SECRET'`	decoded string
`hard_boolean:`	Hardcoded boolean	`'hard_boolean:false'`	`false`
`hard_json:`	Hardcoded JSON	`'hard_json:{"a":1}'`	`{ a: 1 }`
`hard_base64:`	Hardcoded base64	`'hard_base64:SGVsbG8='`	`"Hello"`

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. `.nogit/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
`getKvStore()`	Returns the underlying `KeyValueStore<T>`
`logMissingKeys()`	Logs and returns any required keys that are missing
`waitForAndGetKey(key)`	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.

changelog.md for @push.rocks/smartconfig

2026-03-24 - 6.1.0 - feat(docs)

refresh project documentation and migrate smartconfig tool configuration keys

- rewrites the README with a clearer quick start, updated class documentation, and issue reporting guidance
- updates .smartconfig.json to use scoped tool configuration keys and adds release registry settings
- adds a standalone MIT license file to the repository

2025-08-16 - 5.3.3 - fix(appdata)

Redact sensitive values in AppData logs and add redaction tests

- Add redactSensitiveValue helper in AppData to mask secrets (API keys, tokens, passwords, JWTs, etc.) during logging.
- Use redaction when logging raw and final mapping values in processMappingValue and nested key logging to avoid leaking sensitive data.
- Improve log output for long or special values (JWT/base64 detection, length-aware previews) while preserving actual stored values.
- Add test/test.redaction.ts to verify sensitive environment values are redacted in console output but still stored correctly in the kv store.
- Add local config .claude/settings.local.json (editor/CI permissions/settings).

2025-08-16 - 5.3.2 - fix(dependencies)

Bump @push.rocks/qenv to ^6.1.3 and add local Claude settings

- Bumped dependency @push.rocks/qenv from ^6.1.2 to ^6.1.3 in package.json
- Added .claude/settings.local.json to provide local Claude permissions and tooling allowances for development/CI helpers

2025-08-15 - 5.3.1 - fix(AppData/Conversion)

Improve boolean conversion and mapping evaluation in AppData, ensuring falsy values (like false, 0, and empty strings) are correctly handled and logged. Also, reduce test timeout and add local permissions settings for development.

- Enhanced toBoolean and evaluateMappingValue functions to properly preserve and log falsy values.
- Added detailed logging for mapping spec processing and nested key evaluations.
- Reduced test timeout in package.json for faster CI feedback.
- Introduced .claude/settings.local.json with updated permissions for local development.

2025-08-15 - 5.3.0 - feat(AppData)

Refactor AppData class for declarative env mapping and enhanced static helpers

- Introduced a singleton Qenv provider to optimize environment variable resolution.
- Centralized type conversion logic with utility functions for boolean, JSON, base64, number, and string conversions.
- Replaced complex switch statements with a composable, declarative mapping pipeline for processing envMapping.
- Enhanced logging during AppData initialization to clearly report key processing and overwrite operations.
- Added new static helper methods for environment variable access (valueAsBoolean, valueAsJson, valueAsBase64, valueAsString, valueAsNumber).
- Fixed boolean conversion issues and ensured backward compatibility with the deprecated 'ephermal' option.

2025-08-15 - 5.2.0 - feat(AppData)

Major refactoring of AppData class for improved elegance and maintainability

- **New Features:**
  - Added static helper methods for direct environment variable access:
    - `AppData.valueAsBoolean()` - Convert env vars to boolean
    - `AppData.valueAsJson()` - Parse env vars as JSON
    - `AppData.valueAsBase64()` - Decode base64 env vars
    - `AppData.valueAsString()` - Get env vars as string
    - `AppData.valueAsNumber()` - Parse env vars as number
  - Enhanced logging for AppData initialization and key processing:
    - Shows which storage type is being used (custom, ephemeral, auto-selected)
    - Logs each key being processed with its spec type
    - Reports success/failure for each key with type information
    - Provides summary statistics of processed keys
- **Architecture Improvements:**
  - Replaced 100+ line switch statement with declarative pipeline architecture
  - Introduced centralized type converters and transform registry
  - Implemented composable transform pipeline: `parseMappingSpec()` → `resolveSource()` → `applyTransforms()`
  - Added singleton Qenv provider to reduce allocations
  - Reduced code complexity by ~70% while maintaining 100% backward compatibility
- **Bug Fixes:**
  - Fixed boolean conversion to properly handle both string and boolean inputs
  - Added `ephemeral` option (correctly spelled) while maintaining backward compatibility with deprecated `ephermal`
- **Performance:**
  - Optimized environment variable resolution with shared Qenv instance
  - Reduced object allocations in static helpers

2025-08-15 - 5.1.4 - fix(AppData, dev dependencies, settings)

Improve boolean conversion in AppData, update @types/node dependency, and add local settings file.

- Fixed env var boolean conversion to properly handle non-string values in AppData.
- Updated @types/node from ^20.14.5 to ^22 in package.json.
- Added .claude/settings.local.json to configure project permissions locally.

2025-08-15 - 5.1.3 - fix(appdata)

Fix iteration over overwriteObject in AppData and update configuration for dependency and path handling

- Replaced incorrect looping constructs in the AppData class to properly iterate over overwriteObject keys
- Improved environment variable mapping and file path resolution in multiple TS modules
- Updated dependency versions and adjusted git workflow configurations
- Enhanced project configuration including TS config and build script adjustments

2024-11-06 - 5.1.2 - fix(appdata)

Fix iteration over overwriteObject in AppData class

- Corrected the for loop from in to of inside the AppData class for iterating over overwriteObject keys.

2024-11-05 - 5.1.1 - fix(AppData)

Fix issue with overwrite object handling in AppData class

- Corrected logic to handle cases when overwriteObject is undefined.

2024-11-05 - 5.1.0 - feat(appdata)

Add support for overwriting keys using the overwriteObject option in AppData

- Introduced the overwriteObject option in IAppDataOptions to allow overwriting specific keys in the AppData class.

2024-06-19 - 5.0.17 - 5.0.23 - Core Updates

Routine maintenance and updates to the core components.

- Multiple core updates and fixes improving stability

2024-06-12 - 5.0.13 - 5.0.16 - Core Updates

Maintenance focus on core systems with enhancements and problem resolutions.

- Enhancements and updates in the core functionality

2024-05-29 - 5.0.13 - Documentation Update

Descriptive improvements aligned with current features.

- Updated core description for better clarity in documentation

2024-04-01 - 5.0.10 - Configuration Update

Improved configuration management for build processes.

- Updated `npmextra.json` to reflect changes in git repository management

2024-02-12 - 5.0.0 - 5.0.9 - Major Core Enhancements

A series of critical updates with resolved issues in the core components.

- Introduction of new core features
- Several core system updates

2024-02-12 - 4.0.16 - Major Version Transition

Migration to the new major version with impactful changes.

- BREAKING CHANGE: Significant updates requiring attention for smooth transition

2023-08-24 - 3.0.9 - 4.0.16 - Organization Updates

Formatted updates with attention to organizational standards and practice.

- SWITCH to a new organizational scheme

2023-07-11 - 3.0.9 - Organizational Enhancement

Shifts aligning with contemporary structuring and logistics.

- Strategic realignment with organizational principles