# Data Workspace CLI (CLI)

natural spec v5 - User Manual | schema_version: 2.0 | generated: 2026-06-28 (UTC)

## Overview

The Data Workspace CLI is a deterministic, stdlib-only, multi-command tool for managing local data workspaces, sources, transformations, runs, reports, and configuration. It exposes a subcommand-based interface with strict stdout/stderr discipline, explicit separation between read and write operations, and stable text/JSON output suitable for repeatable workflows.

In practice, you use the CLI to work against a local workspace tree: first initialize and select a workspace, then attach sources, define and run transforms, and track executions as runs. You can later generate summary reports over the stored state or export report artifacts, while configuration commands let you manage profile-scoped settings that control how these operations behave. All state is stored in deterministic JSON files under a configurable workspace root so repeated runs remain predictable.

The CLI entrypoint is `python -m app.cli`, which provides top-level help and a set of command groups: `workspace`, `source`, `dataset`, `transform`, `run`, `report`, and `config`. Global options such as output format, profile selection, workspace root, dry-run, and strict validation are shared consistently across command groups.

## Getting started

Start with the discovery commands below to inspect the available CLI surface.

Use top-level help, command-group help, and action-level help to identify the documented commands, options, and actions.

End-to-end workflows are described only when the available documentary material supports them.

### Starter commands

```
$ python -m app.cli --help
$ python -m app.cli
$ python -m app.cli workspace --help
$ python -m app.cli source --help
$ python -m app.cli transform --help
```

## How to use the prototype

### Discover available commands and global options

Begin by inspecting the top-level help to see all command groups, global options, and the general usage pattern. Each command group also exposes its own help that lists the actions available within that group.

You can safely run these help commands at any time; they are read-only and return exit code 0 when successful.

### Commands

```
$ python -m app.cli --help
$ python -m app.cli workspace --help
$ python -m app.cli source --help
$ python -m app.cli transform --help
$ python -m app.cli run --help
$ python -m app.cli report --help
$ python -m app.cli config --help
```

**Notes**

- Invoking `python -m app.cli` with no arguments prints the same top-level help and exits with code 0.

- All command groups share the same global options: `--format`, `--profile`, `--workspace-root`/`--root`, `--dry-run`, and `--strict`.

## Initialize and manage workspaces

Workspaces provide isolated containers for your data sources, transforms, runs, and reports. Use the `workspace` command group to initialize new workspaces, list existing ones, show details, and delete workspaces you no longer need.

Workspace state is persisted as deterministic JSON under a workspace root directory, which defaults to a dedicated subdirectory when not overridden.

### Commands

```
$ python -m app.cli workspace init
$ python -m app.cli workspace list
$ python -m app.cli workspace show
$ python -m app.cli workspace delete
```

**Notes**

- Use `--workspace-root` or `--root` to change where workspace data is stored on disk.

- `workspace init` and `workspace delete` are mutating operations; combine them with `--dry-run` to see planned effects without writing to disk.

## Manage sources within a workspace

Sources represent local JSON-backed data records associated with a workspace. The `source` command group lets you add sources, list all sources in a workspace, show individual sources, and remove sources.

Source metadata is stored in a per-workspace JSON index, using deterministic formatting and sorted order to keep results stable across runs.

### Commands

```
$ python -m app.cli source add
$ python -m app.cli source list
$ python -m app.cli source show
$ python -m app.cli source remove
```

**Notes**

- Source operations are scoped to a workspace; select the appropriate workspace using `--workspace-root` and any workspace-specific options exposed by the CLI.

- `source add` and `source remove` are mutating operations; `source list` and `source show` are read-only.

- For read commands, stdout contains only the requested record or list (in text or JSON) followed by a single trailing newline.

### Validate, preview, and apply transforms

Transforms define how data from your sources is processed. The `transform` command group offers three main actions: `validate` to check a transform specification, `preview` to see what the transformed data would look like, and `apply` to write actual transformed output to an explicit destination.

Validate and preview operations are read-only and do not change workspace storage. Apply is the only mutating transform operation and writes results to a user-specified target path under the workspace layout.

### Commands

```
$ python -m app.cli transform validate
$ python -m app.cli transform preview
$ python -m app.cli transform apply
```

**Notes**

- Transforms are workspace-scoped and use the same canonical directory layout as sources.

- Use `--dry-run` where supported to see planned apply behavior without writing files.

- You can choose JSON output for previews using `--format json` for machine-friendly inspection.

### Start runs and inspect run status and history

Runs represent tracked executions within a workspace. The `run` command group lets you start a new run, query the status of a specific run, and list the full run history for a workspace.

Run records are stored as separate JSON files under deterministic paths, with monotonically increasing identifiers such as `run-000001`.

### Commands

```
$ python -m app.cli run start
$ python -m app.cli run status
$ python -m app.cli run history
```

**Notes**

- `run start` is a mutating operation that creates a new run record, but it can honor `--dry-run` to return the planned record without persisting it.

- `run status` and `run history` are read-only and return information from existing run files.

- If a requested run is missing, a domain-specific error is emitted to stderr as a single line with a non-zero exit code.

## Generate summaries and export reports

The `report` command group provides summary and export capabilities over workspace state. Use `summary` to build a deterministic aggregate view of sources and runs, and `export` to write a report artifact to a specific output path.

Summary reports are read-only and operate over persisted workspace data. Export is a write-owning operation that creates a declared output file using deterministic JSON or text formatting.

## Commands

```
$ python -m app.cli report summary
$ python -m app.cli report export
```

**Notes**

- `report summary` reads run and source state from the workspace directory and returns a stable aggregate without modifying any files.

- `report export` writes one report artifact to an explicit destination; stdout reports only the completed export action, not the full report content.

- Use `--format json` when you need a machine-readable summary payload on stdout.

## Manage configuration profiles and keys

Configuration is managed via the `config` command group, which is profile-scoped. Use `show` to inspect an entire profile, `get` to read a single configuration key, `set` to update a key, and `reset` to clear or restore configuration.

Configuration data is stored as JSON under canonical profile paths, with deterministic key ordering and formatting.

## Commands

```
$ python -m app.cli config show
$ python -m app.cli config get
$ python -m app.cli config set
$ python -m app.cli config reset
```

**Notes**

- Select the active profile with `--profile`; it defaults to `default` when Not documented.

- `config show` and `config get` are read-only and never create directories.

- `config set` and `config reset` are mutating operations that write JSON configuration under `<base>/config/profiles/` for the selected profile.

## Command reference

### Documented global options

| Global options |
|---|
| --help |
| --format |
| --profile |
| --workspace-root |
| --root |
| --dry-run |
| --strict |

### Group: config

| Field | Value |
|---|---|
| Purpose | Config operations. |
| Actions | show • get • set • reset |
| Notes | Configuration is profile-scoped and persisted as JSON under `<base>/config/profiles/`. • `show` and `get` are read-only; `set` and `reset` write updated configuration mappings with deterministic ordering. |

### Commands

```
$ python -m app.cli config
$ python -m app.cli config --help
```

## Group: dataset

| Field | Value |
| --- | --- |
| Purpose | Dataset operations. |

## Group: report

| Field | Value |
| --- | --- |
| Purpose | Report operations. |
| Actions | summary • export |
| Notes | `summary` reads run and source data from the workspace tree and returns deterministic aggregates. • `export` writes one declared report artifact to a user-specified destination path using deterministic JSON or text. |

## Commands

```
$ python -m app.cli report
$ python -m app.cli report --help
```

## Group: run

| Field | Value |
| --- | --- |
| Purpose | Run operations. |
| Actions | start • status • history |
| Notes | Runs are tracked per workspace as JSON files with monotonically increasing identifiers such as `run-000001`. • `start` supports dry-run behavior to return the planned run record without creating files. |

## Commands

```
$ python -m app.cli run
$ python -m app.cli run --help
```

## Group: source

| Field | Value |
| --- | --- |
| Purpose | Source operations. |
| Actions | add • list • show • remove |

| Field | Value |
| --- | --- |
| Notes | Sources are stored in a per-workspace JSON index; read operations return records sorted by source name. • `add` and `remove` may honor `--dry-run` to avoid persisting changes. |

## Commands

```
$ python -m app.cli source
$ python -m app.cli source --help
```

## Group: transform

| Field | Value |
| --- | --- |
| Purpose | Transform operations. |
| Actions | validate • preview • apply |
| Notes | Transforms are workspace-scoped and share the workspace directory layout with sources. • `validate` and `preview` are read-only; `apply` is mutating and writes only to an explicit target path. |

## Commands

```
$ python -m app.cli transform
$ python -m app.cli transform --help
```

## Group: workspace

| Field | Value |
| --- | --- |
| Purpose | Workspace operations. |
| Actions | init • show • list • delete |
| Notes | Workspace metadata is stored as deterministic JSON under a directory derived from the workspace root. |

## Commands

```
$ python -m app.cli workspace
$ python -m app.cli workspace --help
```

## Inputs and outputs

The CLI operates on local filesystem paths, workspace names, source identifiers, transform specifications, run identifiers, and configuration keys/values. Inputs are typically provided as command-local arguments or options (for example, names, identifiers, and file paths), while shared invocation controls are handled via global options.

Outputs are deterministic text or JSON payloads to stdout for read operations and file-backed JSON or report artifacts for mutating operations. Each successful command emits exactly one payload to stdout followed by a single trailing newline, while errors emit a single line to stderr.

| Type | Value |
| --- | --- |
| Required input | Workspace name for workspace, source, transform, run, report, and related workspace-scoped operations. |
| Required input | Source identifiers and associated record details when adding, showing, or removing sources. |
| Required input | Transform specifications and any referenced source or target file paths for `transform validate`, `preview`, and `apply`. |
| Required input | Run identifiers for `run status` and selection of runs in `run history` where required. |
| Required input | Configuration profile names and configuration keys (and values for updates) for `config show`, `get`, `set`, and `reset`. |
| Required input | Explicit output target paths for write-owning operations such as `report export` and transform apply outputs. |
| Produced output | Workspace index JSON files under the workspace root |
| Produced output | Per-workspace source index JSON files |
| Produced output | Per-run JSON files with monotonically increasing run identifiers |
| Produced output | Deterministic transform validation and preview results in text or JSON |
| Produced output | Summary report payloads in text or JSON |
| Produced output | Exported report files written to user-specified paths |
| Produced output | Profile-scoped configuration JSON files under a config/profiles directory |

## Configuration

Configuration in the CLI is controlled both through global options and through the `config` command group. Global options let you choose the output format, active configuration profile, workspace root directory, and runtime behavior flags such as dry-run and strict validation.

Profile-scoped configuration values are persisted as JSON files under a canonical directory layout. You use `config show`, `get`, `set`, and `reset` to inspect and modify these values for each profile.

| Configuration principles |
| --- |
| `--format {text,json}` controls whether command outputs are rendered as human-readable text or as deterministic JSON. |
| `--profile PROFILE` selects the active configuration profile, defaulting to `default` when Not documented. |
| `--workspace-root WORKSPACE_ROOT` (alias `--root`) sets the base directory for workspace data and related JSON files. |
| `--dry-run` runs mutating commands without applying changes, returning planned results only. |
| `--strict` enables stricter validation behavior for command inputs. |
| Configuration profiles are stored as JSON files under `<base>/config/profiles/`, with deterministic key ordering and formatting. |

## Current limitations

The prototype is designed around deterministic local filesystem operations and JSON-backed storage. It intentionally focuses on a constrained feature set suitable for predictable, repeatable workflows rather than broad integration capabilities.

| Documented limitations |
| --- |
| Operations are limited to local filesystem paths; no remote storage or network integration is documented. |
| Only `text` and `json` output formats are supported via the `--format` option. |
| Persistent state for workspaces, sources, runs, reports, and configuration is stored exclusively as JSON files using deterministic formatting. |
| Write-owning operations such as transform apply and report export require explicit target paths; there is no implicit or default export destination beyond the documented layout. |

## Project map

### Application layout

```
app/
|_ cli.py
|_ orchestrator.py
|_ config_store.py
|_ fs_paths.py
|_ json_formatter.py
|_ reporting.py
|_ run.py
|_ source.py
|_ storage.py
|_ transform.py
|_ validators.py
|_ workspace.py
```