ADAC CLI

adac-cli is the command-line interface for working with **ADAC (Archival Digital Asset Container)** files. It exposes the core capabilities of the `Adac` library — inspecting, validating, verifying, extracting, and creating `.adac` containers — directly from the terminal.

Table of Contents

↑ Back to Top

Prerequisites

- .NET 10 SDK or later.

↑ Back to Top

Installation

Building from source

Clone the repository and build the CLI project:

```
git clone https://github.com/InnoVadens/Adac.git
cd Adac
dotnet build Adac.Cli/Adac.Cli.csproj
```

Or build the entire solution:

```
dotnet build
```

Publishing a self-contained binary

To produce a single executable that does not require the .NET SDK at runtime:

```
dotnet publish Adac.Cli/Adac.Cli.csproj -c Release -r win-x64 --self-contained
```

Replace `win-x64` with your target runtime identifier (e.g. `linux-x64`, `osx-arm64`).

↑ Back to Top

Quick Start

```
# Create a container from two master TIFF files
adac-cli create output.adac --master scan_001.tif --master scan_002.tif --title "Lette

# Inspect what's inside
adac-cli info output.adac

# List every entry
adac-cli list output.adac

# Validate against the ADAC 1.0 specification
adac-cli validate output.adac

# Verify file integrity checksums
adac-cli verify output.adac

# Extract everything to a directory
adac-cli extract output.adac ./extracted
```

↑ Back to Top

Usage

Run from the project directory:

```
dotnet run --project Adac.Cli -- <command> [arguments] [options]
```

After publishing or installing as a tool, invoke directly:

```
adac-cli <command> [arguments] [options]
```

Global Options

Option	Description
`--version`	Show version information.
`-h`, `--help`	Show help and usage information.

Every command also accepts `-h` / `--help` to display its own usage details.

```
adac-cli --help
adac-cli create --help
```

↑ Back to Top

Commands

`info` — Display Container Metadata

Reads and prints the **manifest** (`manifest.json`) and **core metadata** (`metadata/core.json`) from an ADAC container as formatted JSON.

```
adac-cli info <file>
```

Argument	Description
`<file>`	Path to the `.adac` container.

Example:

```
adac-cli info archive.adac
```

Sample output:

```
=== Manifest ===
{
  "adacVersion": "1.0",
  "id": "b3f1a2c4-...",
  "createdOn": "2025-07-15T12:00:00+00:00",
  "createdBy": "adac-cli/1.0.0",
  "description": "Scanned photograph",
  "masters": [ ... ]
}

=== Core Metadata ===
{
  "id": "b3f1a2c4-...",
  "title": "Family portrait, 1923",
  "format": "TIFF"
}
```

The manifest section shows the ADAC version, unique container ID, creation timestamp, and the list of master files. The core metadata section shows descriptive cataloging information such as title and format.

`list` — List Container Entries

Lists every entry (file path) inside the ADAC container's ZIP archive.

```
adac-cli list <file>
```

Argument	Description
`<file>`	Path to the `.adac` container.

Example:

```
adac-cli list archive.adac
```

Sample output:

```
Entries (6):
  manifest.json
  metadata/core.json
  master/master_0001.tif
  provenance/log.json
  provenance/checksums.json
  metadata/xmp/master_0001.xmp
```

Use this command to get a quick overview of everything stored in the container before extracting or validating.

`validate` — Validate Container Structure

Validates an ADAC container against the ADAC 1.0 specification. Reports errors, warnings, and informational findings. By default, stored checksums are also verified; use `--skip-checksums` to perform structural validation only.

```
adac-cli validate <file> [--skip-checksums]
```

Argument / Option	Description
`<file>`	Path to the `.adac` container.
`--skip-checksums`	Skip SHA-256 checksum verification during validation.

Example — full validation (structure + checksums):

```
adac-cli validate archive.adac
```

Example — structure only (faster):

```
adac-cli validate archive.adac --skip-checksums
```

Sample output:

```
  [INFO] ADAC-010: Container created with ADAC version 1.0.
  [WARN] ADAC-040: Provenance log is missing. (provenance/log.json)
  [ERR ] ADAC-020: Master directory is empty. (master/)

Container is INVALID.
```

Each finding includes:

Field	Meaning
Severity	`INFO` (informational), `WARN` (non-fatal issue), or `ERR` (spec violation).
Code	A stable identifier (e.g. `ADAC-010`) useful for filtering or scripting.
Message	Human-readable description of the finding.
Path	The container-relative path involved, if applicable.

The final line prints **`Container is VALID.`** when no error-level findings exist, or **`Container is INVALID.`** otherwise.

`verify` — Verify Fixity Checksums

Computes SHA-256 hashes for every file in the container and compares them against the stored checksum manifest (`provenance/checksums.json`). This is the integrity check archivists use to confirm that no content has been silently altered or corrupted since the container was created.

```
adac-cli verify <file>
```

Argument	Description
`<file>`	Path to the `.adac` container.

Example:

```
adac-cli verify archive.adac
```

Sample output (all passing):

```
Total files:    5
Verified:       5
Failed:         0
Missing:        0

Fixity check PASSED.
```

Sample output (with failures):

```
Total files:    5
Verified:       3
Failed:         1
Missing:        1

Mismatches:
  MISMATCH master/master_0001.tif
           expected: a1b2c3d4...
           actual:   ff00ff00...
  MISSING  metadata/xmp/master_0001.xmp

Fixity check FAILED.
```

`extract` — Extract Files from a Container

Extracts content from an ADAC container to a directory on disk. Supports extracting everything, only masters, a single master by ID, or a single entry by its container path.

```
adac-cli extract <file> <output> [--entry <path>] [--master <id>] [--masters-only]
```

Argument / Option	Description
`<file>`	Path to the `.adac` container.
`<output>`	Destination directory for extracted files. Created if it does not exist.
`--entry <path>`	Extract a single entry by its container-relative path (e.g., `metadata/core.json`).
`--master <id>`	Extract a single master file by its identifier (e.g., `master_0001`).
`--masters-only`	Extract only the master files.

When none of the filtering options are provided, **all entries** are extracted with their original directory structure preserved.

Examples:

```
# Extract everything
adac-cli extract archive.adac ./output

# Extract only master files
adac-cli extract archive.adac ./masters --masters-only

# Extract a specific master by ID
adac-cli extract archive.adac ./output --master master_0001

# Extract a single entry by path
adac-cli extract archive.adac ./output --entry metadata/core.json
```

Sample output (all entries):

```
  manifest.json
  metadata/core.json
  master/master_0001.tif
  provenance/log.json
  provenance/checksums.json
Extracted 5 entries to C:\output
```

`create` — Create a New ADAC Container

Creates a new `.adac` container from one or more master files on disk. The CLI generates a valid manifest, core metadata, checksum manifest, and the required directory structure automatically.

```
adac-cli create <output> --master <path> [--master <path> ...] [options]
```

Argument / Option	Description
`<output>`	File path for the new `.adac` container.
`--master <path>`	**(Required)** Path to a master file to include. Repeat for multiple masters.
`--title <text>`	Title of the artifact (written to core metadata).
`--description <text>`	Human-readable description (written to the manifest and core metadata).
`--created-by <text>`	Software or person creating the container. Defaults to `adac-cli/<version>`.

Examples:

```
# Single master
adac-cli create photo.adac --master scan.tif --title "Family portrait, 1923"

# Multiple masters with description
adac-cli create collection.adac \
  --master page_001.tif \
  --master page_002.tif \
  --description "Two-page letter, dated 1945" \
  --title "Wartime correspondence" \
  --created-by "Digitization Lab v2"
```

Sample output:

```
  [CopyingMasters] (1/2) page_001.tif
  [CopyingMasters] (2/2) page_002.tif
  [WritingChecksums] (1/1) provenance/checksums.json
Created: C:\archives\collection.adac
```

Master files are assigned sequential identifiers (`master_0001`, `master_0002`, …) and stored uncompressed in the `master/` directory, following the ADAC 1.0 specification's preservation-first principle.

↑ Back to Top

Typical Workflows

Ingest and verify a new scan

```
# Package the master file
adac-cli create scan.adac --master original.tif --title "Deed of sale, 1887"

# Confirm the container is well-formed
adac-cli validate scan.adac

# Later — verify nothing changed in transit
adac-cli verify scan.adac
```

Inspect an existing container

```
# What's inside?
adac-cli list archive.adac

# Show full metadata
adac-cli info archive.adac
```

Extract masters for processing

```
adac-cli extract archive.adac ./working-copies --masters-only
```

Batch-validate a folder of containers

On Linux / macOS:

```
for f in /archive/*.adac; do
  echo "--- $f ---"
  adac-cli validate "$f" --skip-checksums
done
```

On Windows (PowerShell):

```
Get-ChildItem C:\archive\*.adac | ForEach-Object {
    Write-Host "--- $($_.Name) ---"
    adac-cli validate $_.FullName --skip-checksums
}
```

↑ Back to Top

Exit Codes

Code	Meaning
`0`	Command completed successfully.
`1`	A runtime error occurred (e.g., file not found, invalid input).

> **Note:** `validate` and `verify` currently return exit code `0` even when the container is invalid or fixity checks fail. Check the textual output (`Container is VALID.` / `INVALID.`, `Fixity check PASSED.` / `FAILED.`) to determine the result programmatically.

↑ Back to Top

Error Handling

When a command encounters an error, a message is written to **stderr** and the process exits with a non-zero code.

Common error scenarios:

Scenario	Output
Container file does not exist	`File not found: /path/to/file.adac`
Master file does not exist (during `create`)	`Master file not found: /path/to/scan.tif`
Entry not found in container (during `extract --entry`)	Error message from the underlying library

All user-facing error messages are prefixed with enough context to identify the problematic file or argument.

↑ Back to Top

Project Structure

```
Adac.Cli/
├── Adac.Cli.csproj           # Console project (net10.0)
├── Program.cs                 # Entry point — wires up the root command
├── CliServiceProvider.cs      # Builds the DI container with ADAC services
└── Commands/
    ├── InfoCommand.cs         # info
    ├── ListCommand.cs         # list
    ├── ValidateCommand.cs     # validate
    ├── VerifyCommand.cs       # verify
    ├── ExtractCommand.cs      # extract
    └── CreateCommand.cs       # create
```

All commands use the `Adac` core library via dependency injection (`AddAdacCore()`). The CLI itself is stateless — each invocation creates a fresh service provider and disposes it on completion.

↑ Back to Top

License

↑ Back to Top