Blender API Capsule Extractor — First Backend for ARCHCode 3D

User Manual | schema_version: 2.0 | generated: 2026-06-29 (UTC)

Overview

Blender API Capsule Extractor is a local, deterministic, stdlib-only CLI for building backend documentation capsules for ARCHCode 3D. It reads an already-unpacked official Blender Python API HTML documentation tree and produces artifacts focused on the relationships.py backend, without requiring Blender itself.

The tool works in two main modes. In build mode, it walks the Blender API HTML tree under a given root, extracts documentation for a small set of required symbols, and assembles a JSON capsule plus a readable text version at explicit output paths. In inspect mode, it focuses on a single logical symbol, resolving and parsing its documentation pages and printing diagnostics (and optionally related optional pages) without writing capsule files.

All work happens locally: the CLI reads only local HTML files, never reaches the network, and does not execute Blender. It is intended for ARCHCode developers who need repeatable, explicit artifacts that help ACW project ARCHCode 3D spatial relationships into Blender Python code, while treating Blender as just one backend.

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 build --help
$ python -m app.cli inspect --help
```

How to use the prototype

Discover available commands and options

Use the built-in help to understand the CLI surface, including the build and inspect workflows and their required options. Running the CLI with no arguments or with --help shows the same top-level usage information.

Subcommand help shows the exact required and optional flags for each workflow, which you should review before running real builds or inspections.

Commands

```
$ python -m app.cli
$ python -m app.cli --help
$ python -m app.cli build --help
$ python -m app.cli inspect --help
```

Notes

- The no-args invocation and the top-level --help invocation both print the same usage information and exit with status code 0.

- Each subcommand has its own -h/--help flag that shows required options, optional flags, and brief descriptions.

Build JSON and text capsule artifacts from the Blender API HTML tree

Use the build command when you want to generate persistent capsule artifacts for relationships.py. You must point the tool at the root of your local Blender Python API HTML documentation tree and provide explicit output paths for both a JSON capsule and an ACW-readable text capsule.

Optional flags let you enable strict validation of required pages and capsule sections, or run in dry-run mode to see what would be resolved without writing any files.

Commands

```
$ python -m app.cli build --api-html-root /path/to/blender_api_html
     --out-json /path/to/capsule.json --out-text /path/to/capsule.txt
$ python -m app.cli build --api-html-root /path/to/blender_api_html
     --out-json /path/to/capsule.json --out-text /path/to/capsule.txt --strict
$ python -m app.cli build --api-html-root /path/to/blender_api_html
     --out-json /path/to/capsule.json --out-text /path/to/capsule.txt --dry-run
```

Notes

- --api-html-root is required and must point to the local root of the official Blender Python API HTML documentation tree.

- --out-json and --out-text are both required; build expects explicit output paths for the JSON backend capsule artifact and the ACW-readable text capsule artifact.

- --strict enables strict mode for required pages and capsule sections, which may surface additional validation errors.

- --dry-run reports intended resolutions without writing any artifacts to disk.

Inspect a single Blender API symbol and view diagnostics

Use the inspect command when you want to focus on one logical symbol, such as bpy.types.Object, and understand how its documentation is resolved and parsed. This workflow does not write capsule artifacts; instead, it reports diagnostics to stdout.

You must provide the API HTML root and the logical symbol name. You can optionally supply a local Blender manual HTML root, choose to include optional symbol pages in diagnostics, and enable strict mode or dry-run behavior.

Commands

```
$ python -m app.cli inspect --api-html-root /path/to/blender_api_html
     --symbol bpy.types.Object
$ python -m app.cli inspect --api-html-root /path/to/blender_api_html
     --symbol bpy.types.Object --include-optional --strict
$ python -m app.cli inspect --api-html-root /path/to/blender_api_html
     --manual-html-root /path/to/blender_manual_html --symbol bpy.types.Collection
     --dry-run
```

Notes

- --api-html-root is required for inspect, just as for build.

- --symbol is required and must be a logical symbol name such as bpy.types.Object.

- --manual-html-root is optional and allows you to supply a local Blender manual HTML root for additional context.

- --include-optional controls whether optional symbol pages are included in inspect diagnostics.

- --strict and --dry-run behave similarly to build: strict tightens validation, while dry-run reports intended resolutions without writing artifacts.

Command reference

Documented global options

Global options
--help

Group: build

Field	Value
Purpose	Build capsule artifacts from the official Blender Python API HTML documentation.
Notes	--api-html-root specifies the path to the local root of the official Blender Python API HTML documentation tree. • --out-json is the output path for the JSON backend capsule artifact. • --out-text is the output path for the ACW-readable text capsule artifact. • --strict enables strict mode for required pages and capsule sections. • --dry-run reports intended resolutions without writing artifacts.

Commands

```
$ python -m app.cli build --help
$ python -m app.cli build --api-html-root API_HTML_ROOT --out-json OUT_JSON
     --out-text OUT_TEXT [--strict] [--dry-run]
```

Group: inspect

Field	Value
Purpose	Inspect a single symbol and report diagnostics based on the Blender API documentation.
Notes	--api-html-root specifies the path to the local root of the official Blender Python API HTML documentation tree. • --manual-html-root is an optional path to the local Blender manual HTML root. • --symbol is the logical symbol to inspect, for example bpy.types.Object. • --include-optional includes optional symbol pages in inspect diagnostics. • --strict enables strict mode for required pages and capsule sections. • --dry-run reports intended resolutions without writing artifacts.

Commands

```
$ python -m app.cli inspect --help
$ python -m app.cli inspect --api-html-root API_HTML_ROOT [--manual-html-root
    MANUAL_HTML_ROOT] --symbol SYMBOL [--include-optional] [--strict] [--dry-run]
```

Inputs and outputs

The CLI consumes local HTML documentation trees and command-line parameters, and it produces JSON and text capsule artifacts for build workflows, or diagnostics printed to stdout for inspect workflows.

All file I/O uses explicit user-supplied paths. The tool does not discover files outside the provided roots, and it does not access the network or require a running Blender instance.

Type	Value
Required input	Path to the local root of the official Blender Python API HTML documentation tree (--api-html-root).
Required input	Output path for the JSON backend capsule artifact (--out-json) when running the build command.
Required input	Output path for the ACW-readable text capsule artifact (--out-text) when running the build command.
Required input	Logical symbol name to inspect, such as bpy.types.Object (--symbol), when running the inspect command.
Produced output	JSON backend capsule artifact for relationships.py.
Produced output	ACW-readable text capsule artifact for relationships.py.
Produced output	Inspect diagnostics for a single symbol printed to stdout.

Configuration

This prototype is configured entirely through command-line options. There is no separate configuration file or persistent profile to edit.

You control which HTML trees are used, where artifacts are written, and how strict the validation should be by choosing appropriate flags for the build and inspect commands.

Current limitations

The prototype is intentionally narrow in scope and focuses on producing backend capsules for relationships.py from the official Blender Python API documentation. It does not attempt to be a general-purpose Blender documentation tool.

Its operation is limited to local files and an already-unpacked Blender HTML tree, and it targets a fixed set of required API symbols for the Blender backend capsule.

Documented limitations
Runs only as a local CLI and reads only local HTML files; no network access is performed.
Does not require or interact with a running Blender runtime.
Targets Blender as the first backend for ARCHCode 3D and focuses on the relationships.py role rather than arbitrary modules.
Requires specific official API symbol pages (including bpy.types.Object, bpy.types.Collection, and bpy.types.ID) to satisfy the minimum build contract.
Certain symbols such as mathutils.Vector, mathutils.Matrix, mathutils.Euler, and mathutils.html pages are excluded from the minimum viable build contract.
Writes artifacts only to explicit output paths supplied on the command line; no implicit output locations are used.

Project map

Application layout

```
app/
|_ cli.py
|_ orchestrator.py
|_ capsule_builder.py
|_ contracts.py
|_ fs_io.py
|_ html_extraction.py
|_ path_resolution.py
|_ profile.py
|_ rendering.py
```