Curio CLI

Installation, Usage, and AI Agents

Zengobi, Inc.

Table of contents

Installing the CLI

Curio supports automation through a separately downloaded command line tool named `curio`. The CLI can inspect and modify Curio projects from Terminal, scripts, and AI agents.

See what you can build
Visit the Curio CLI overview to explore the CLI Examples, with downloadable recipes and practical tutorials for dashboards, AI-agent workflows, and more.

Availability and Updates
1️⃣ The Curio CLI is an extra perk for Curio Professional sustainers with either an active Mac App Store subscription or a traditional Curio license within its current update window. If your subscription or update window has lapsed, renew from the appropriate online store.
2️⃣ You are also expected to keep current with Curio CLI and app releases so we can ship timely security and data-integrity fixes, protocol updates, bug fixes, and new features. This is enforced: see Updating the CLI below.

What the CLI Does

The `curio` command gives you a second way into your projects. Anything you would normally do by clicking around a project—finding figures, reading and changing their content, filing them into Organizer sections, bringing content in and sending it back out—you can also do with a command, from a script, or through an AI agent:

- **Query.** Search a project with Curio's query language, spanning text, tags, dates, status, ratings, flags, and metadata.
- **Read and update.** Inspect and change titles, text, checkboxes, tags, dates, meta values, figure geometry, and dozens of other figure and Organizer item fields.
- **Organize.** Create Organizer sections, folders, and idea spaces, then move figures and Organizer items where they belong.
- **Import and export.** Bring files, text, and structured content into a project, spread PDF pages across idea spaces, and export figures and idea spaces back out.
- **Inspect.** List the Organizer hierarchy, project assets, available tags, available fields, Project Center categories, and the state of the running app.
- **Navigate.** Send the Curio app to a specific figure or idea space, open deep links, or work with whatever is currently selected.

There are two ways to work: by default the CLI talks to the running Curio app, which supports the full command set including selection-aware reads and every write operation. You can also point it at a `.curio` file directly with `--project`, which reads the project from disk without launching Curio—useful for read-only scripts and scheduled jobs.

Results print as readable text for people, or as JSON with meaningful exit codes for scripts and agents. See JSON Output and Exit Codes.

For AI, the CLI ships bundled skill files that teach command-line agents such as Codex and Claude Code how to drive the `curio` command, plus a built-in MCP server so MCP hosts such as ChatGPT Desktop, Claude Desktop, and Antigravity can work with your Curio data directly.

Downloading and Installing

The Curio CLI is a separate download from the main Curio application. Installing the Curio app does **not** automatically install the `curio` command line tool.

The Curio CLI installer is code-signed with our Apple Developer ID certificate and notarized by Apple, which means macOS has verified that it comes from Zengobi and is free of known malicious content.

You can install or update the CLI from within Curio using the **Curio > Install Curio CLI** or **Curio > Update Curio CLI** menu item.

- In the website edition, Curio downloads the signed installer, verifies it, reveals it in Finder, and opens it.
- In the Mac App Store edition, Curio opens the CLI web page so you can download and install it yourself.

Alternatively, download the latest CLI release directly from zengobi.com/curio/cli/download.

Verifying the Install

After installing it, open Terminal and run:

```
curio --version
```

The signed installer places the release payload like this:

```
/
|-- Library/
|   `-- Application Support/
|       `-- CurioCLI/
|           |-- bin/
|           |   |-- curio
|           |   `-- Frameworks/
|           |-- skills/
|           |   |-- curio/
|           |   |-- curio-query/
|           |   `-- curio-mcp/
|           `-- resources/
|               `-- MCPB/
`-- usr/
    `-- local/
```

```
        `-- bin/
            `-- curio -> /Library/Application Support/CurioCLI/bin/curio
```

The CLI stores its update cache in `~/Library/Application Support/CurioCLI/config.json`. User-facing CLI preferences live separately in `~/.curio/settings.json`; see Settings.

If Terminal still reports that `curio` cannot be found, make sure the installer completed successfully and verify that `/usr/local/bin/curio` exists.

Updating the CLI

> **Important**
> To ensure robust handling of your project data, you are expected to keep up with both Curio CLI releases and Curio app releases.

To review what changed in recent CLI releases, see the Curio CLI release notes.

The CLI checks for updates automatically once per day before handling normal commands such as `query` and `get`. The first such command of the day fetches the latest CLI release metadata and caches it locally; later commands that same day reuse that cached result.

If a newer CLI release is available, the CLI prints an update notice to standard error before handling the request and tells you how many days remain before the update becomes mandatory. For the first 30 days after a newer CLI release is published, the command still runs. After that 30-day grace period, the CLI stops the request and returns an error until you upgrade.

Use this command at any time to refresh the update state immediately:

```
curio update
```

If an update is available and the CLI was installed with the signed installer, `curio update` asks whether to download and launch the installer. Press `y` or Return to download the installer to your Downloads folder, reveal it in Finder, and open it. Press any other key to leave the installer untouched.

Before opening a downloaded installer, the CLI verifies the package against the SHA-256 checksum in the update metadata, checks its installer signature with macOS `pkgutil`, requires the Zengobi Developer ID Installer identity, and runs a Gatekeeper installer assessment with `spctl`. If any verification step fails, the CLI removes the downloaded package and does not open it.

When `curio update` is run from a script, through redirected input/output, or through an MCP host, it does not prompt; it only reports the update details and download URL.

Some commands remain available even when an update is required: `curio docs`, `curio help`, `curio --help`, `curio --version`, `curio fields`, `curio skills`, `curio settings`, `curio claude`, and `curio update`.

Each CLI release also declares the minimum Curio app version it requires. If your installed Curio app is too old, the CLI will refuse the request and tell you which Curio version is required.

Next Steps

- Read Using the CLI for querying projects, reading and updating content, importing and exporting, navigation, settings, command output, and troubleshooting.
- Read AI Agents to install Curio skills or connect ChatGPT, Codex, Claude, Antigravity, and other MCP hosts.

Using the CLI

This guide covers using the Curio CLI to inspect and modify Curio projects, organize content, import and export files, navigate the Curio app, and work with command output and errors.

Two Ways to Work

By default, `curio` talks to the running Curio app. This mode supports everything, including selection-based queries, Organizer creation, property changes, import/export, and navigation by UUID.

```
curio query "#active" --fields id,title
```

You can also point the CLI at a project file directly using `--project`:

```
curio query --project ~/Documents/MyProject.curio "#active" --fields id,title
```

This direct mode is read-only, so it is ideal for scripts and automations that need to inspect a project without launching Curio first.

When several projects are open, add `--project-id <projectUUID>` to target one of them without bringing its window forward:

```
curio query --project-id 12345678-1234-1234-1234-123456789ABC "#active" --fields id,title
```

Get the UUID from `curio status`. Live project targeting is supported by `query`, `get`, `set`, `tags`, `create`, `move`, `batch`, `export`, `import`, `navigate`, `save`, `sections`, and `assets`. It establishes the project context once for the request; only `navigate` activates the targeted project. Do not combine `--project-id` with the offline `--project <path>` option.

> **Important**
> `--project` mode is read-only. You can use it with `query`, `get`, `sections`, `assets`, and `tags`, but not `project`, `set`, `create`, `export`, `import`, `navigate`, `project-center`, or `status`.

> **Note**
> Direct project access requires a recent activation token. If you see a `NotActivated` error, simply launch the Curio app once and try again.

Finding Help

The CLI is self-documenting. These commands are the quickest way to inspect the capabilities of the version you have installed:

```
curio help
curio --help
curio docs
curio fields
curio fields due_date
curio tags
curio settings list
curio skills list
curio claude doctor
curio mcp-server --log ~/Library/Logs/curio-mcp.log
curio status
curio --version
```

Use `--verbose` with most commands if you want diagnostic details about transport selection, repository discovery, or activation handling.

Quick Start

Check whether Curio is running:

```
curio status
```

Query the running app using Curio's full query language:

```
curio query "#active due soon sort:priority" --fields id,title,priority,due_date
```

Read the currently selected figure:

```
curio get --selected figure --fields plain_text,priority
```

Read a specific figure:

```
curio get --id 12345678-1234-1234-1234-123456789ABC --fields raw_text,due_date,tags,asset_file,sync_file
```

Get the selected idea space:

```
curio get --selected ideaspace --fields title,note,tags
```

List all available tags in the current project:

```
curio tags
curio tags --project ~/Documents/MyProject.curio
```

Inspect Project Center categories and project paths:

```
curio project-center categories
curio project-center projects --category "Active"
```

Create and open a completely empty project:

```
curio project create ~/Documents/NewProject
```

Inspect the project's section hierarchy:

```
curio sections --project ~/Documents/MyProject.curio
```

List assets under a section or Organizer parent:

```
curio assets --project ~/Documents/MyProject.curio --section 12345678-1234-1234-1234-123456789ABC --recursive --organizer_
curio assets --project ~/Documents/MyProject.curio --parent 87654321-4321-4321-4321-CBA987654321 --recursive --organizer_o
```

Create Organizer structure in the running app:

```
curio create section --title "Research"
curio create folder --section 12345678-1234-1234-1234-123456789ABC --title "Exhibits"
curio create ideaspace --section 12345678-1234-1234-1234-123456789ABC --title "001 - Complaint"
curio create ideaspace --selected ideaspace --position next_sibling --title "002 - Exhibit A"
```

Move a figure on the canvas or move an idea space in the Organizer:

```
curio move --selected figure --position 100,100
curio move --id 12345678-1234-1234-1234-123456789ABC --relative-to 87654321-4321-4321-4321-CBA987654321 --position next_si
curio move --id 12345678-1234-1234-1234-123456789ABC --parent 87654321-4321-4321-4321-CBA987654321 --position first_child
```

Update the selected figure:

```
curio set --selected figure --priority 4 --due-date 2026-04-01
curio set --selected figure --url "apple.com"
curio set --selected figure --field priority=3 --values '{"note":"Reviewed"}'
curio set --selected figure --progress 60 --checkmark-visible true
curio set --selected figure --field size.width=360
curio set --selected figure --field fill_color=#ffcc66 --field pen_color=#112233
```

Send text through standard input:

```
echo "hello from Terminal" | curio set --selected figure --raw-text --stdin
```

Set an asset figure caption:

```
curio set --selected figure --caption "Sprint planning whiteboard"
```

Update the selected idea space:

```
curio set --selected ideaspace --title "Roadmap" --tags "GTD/Active,Planning"
curio set --selected ideaspace --field dimension.width=1600 --field dimension.height=1000 --field fill_color=#f0f7ff
```

Export the currently selected figures as PDF:

```
curio export --selected figures --format pdf --filename ~/Desktop/Selection
```

Import a file into the current idea space:

```
curio import ~/Photos/diagram.png
```

Import content as a structured figure:

```
curio import ~/outline.md --as list
curio import ~/outline.md --as list --ignore-inline-tags --ignore-inline-resources
```

Spread a multi-page PDF into one idea space per page:

```
curio import ~/Documents/affidavit.pdf --as spread-pdf --section <sectionUUID> --title-template "{page:03}"
curio import ~/Documents/affidavit.pdf --as spread-pdf --section <sectionUUID> --layout layout.json
```

Command Summary

Command	Purpose
`curio query ...`	Query figures using Curio's query language
`curio get ...`	Read fields from one figure or one Organizer item
`curio sections ...`	Return the Organizer section hierarchy
`curio assets ...`	List project assets or assets within a section
`curio project create ...`	Create and open an empty project with a Default Section
`curio project-center ...`	Inspect Project Center categories and project paths
`curio create ...`	Create Organizer sections, folders, and idea spaces
`curio move ...`	Move a figure on the canvas or move an Organizer item
`curio batch ...`	Run a command batch file with pseudo-variable references
`curio set ...`	Update writable fields on one figure or one Organizer item
`curio export ...`	Export a figure, selected figures, or selected idea spaces
`curio import ...`	Import a file/content or Spread PDF pages
`curio navigate ...`	Open a deep link or project, or navigate to an item by UUID or identifier
`curio save ...`	Synchronously save the current project or an open project UUID
`curio status`	Report whether Curio is running, open projects, and sandbox info
`curio skills`	Manage AI platform skill installations
`curio claude`	Install, package, or diagnose the Claude Desktop connector
`curio mcp-server`	Run the stdio MCP server for host-launched AI integrations
`curio fields`	List all available fields
`curio fields <name>`	Show details for one field
`curio tags`	List available tags, or create project/global tags and tag sets
`curio update`	Check for the latest CLI release
`curio docs`	Open the detailed CLI documentation in the default browser
`curio --help`	Show built-in command help
`curio --version`	Show the installed CLI version

Status

The `status` command reports whether Curio is running, which projects are open, and whether the app is sandboxed:

```
curio status
```

When Curio is running as the Mac App Store edition, the response includes `app_sandboxed: true` and an `accessible_folders` array listing the folders the sandboxed app can read from:

```
{
  "api_version": 1,
  "app_version": "26.1",
  "response_ok": true,
```

```
  "result": {
    "accessible_folders": [
      "/Users/you/Library/Containers/com.zengobi.curio/Data/tmp/",
      "/Users/you/Library/Containers/com.zengobi.curio/Data/Downloads",
      "/Users/you/Documents/Curio",
      "/Users/you/Desktop"
    ],
    "app_open_projects": [
      { "active": true, "has_unsaved_changes": false, "path": "/Users/you/Documents/Curio/MyProject.curio", "title": "MyPr
    ],
    "app_running": true,
    "app_sandboxed": true
  }
}
```


The accessible folders typically include:

- The app's **temporary directory** (inside the sandbox container, not system `/tmp`)
- The app's **Downloads directory** (also inside the container)
- Your **Curio projects folder** (set in Preferences)
- Any **authorized folders** you have previously granted Curio access to

When the website (direct download) edition is running, `app_sandboxed` is `false` and `accessible_folders` is not included — the app can access any file path.

Each entry in `app_open_projects` includes `active` and `has_unsaved_changes` booleans. Exactly one open project is active, so scripts can determine which project receives live commands that depend on Curio's current UI state. `has_unsaved_changes` reports whether the project has edits that have not reached disk. Status settles pending model propagation before reporting this value. Use `curio get --selected ideaspace --fields id,title,section_path` to inspect the active project's current idea space.

When Curio is not running, `status` returns a local response with `app_running: false`.

Creating a Project

Use `project create` to create and open a completely empty Curio project. The project contains its normal initial **Default Section**, but no folders, idea spaces, or figures:

```
curio project create ~/Documents/Test
```

The command requires the running Curio app. The destination must be an absolute path; paths beginning with `~/` are expanded, and a missing `.curio` extension is added automatically. An explicitly different extension is rejected. The parent folder must already exist, and Curio refuses to overwrite an existing file, project, or symbolic link.

A successful result includes `project_uuid`, `path`, `title`, `default_section_id`, and `default_section_title`. The new project is open and active when the command returns, so the returned section UUID can be passed directly to existing Organizer creation commands:

```
curio project create ~/Documents/Test
curio create folder --section <default_section_id> --title "Research"
curio create ideaspace --section <default_section_id> --title "Notes"
curio save --project-id <project_uuid>
```

Project Center

The `project-center` command reads Curio's Project Center categories and tracked-project paths. When Curio is running, it uses the freshest live Project Center model. Otherwise, it reads the last persisted Project Center snapshot and the separate Recently Opened bookmark list without launching Curio or modifying either file.

Every successful result includes `data_source`, which is `live` or `persisted`. A persisted response reflects the most recently saved Project Center state; changes that have not yet been written by the running app require the live path.

List the visible categories:

```
curio project-center categories
```

Category results include:

- `id`: stable category UUID
- `title`: displayed category title
- `kind`: `all`, `recent`, or `category`
- `project_count`: number of projects currently in the category
- `packaged`: whether Curio supplied the category
- `sort_by_title`: whether the category's projects are title-sorted

The result includes the special **All Categorized Projects** and **Recently Opened** categories in both live and persisted modes. Use their returned IDs rather than localized titles when saving a dashboard selection.

List projects in a category by UUID or case-insensitive title:

```
curio project-center projects --category <category-uuid>
curio project-center projects --category "Active"
```

Omit `--category` to list All Categorized Projects:

```
curio project-center projects
```

Project results include `title`, `path`, `valid`, `status_ignored`, and category memberships. Entries without either a resolved path or a last-known path are omitted. A persisted project may also include `path_is_last_known` when its bookmark could not be resolved. Use the path with offline read commands to inspect the project without opening it:

```
curio query --project "/Users/you/Documents/Business.curio" "#meta/unchecked sort:due" --fields id,title,due_date,priority
```

Navigating

Use `navigate` to jump to a Curio location:

```
curio navigate "curio://project/MyProject.curio?figure=abc-123"
curio navigate ~/Documents/MyProject.curio
curio navigate --id 12345678-1234-1234-1234-123456789ABC
curio navigate --project-id ABCDEFAB-1234-5678-90AB-ABCDEFABCDEF
curio navigate --project-id ABCDEFAB-1234-5678-90AB-ABCDEFABCDEF --id 12345678-1234-1234-1234-123456789ABC
```

The positional item form remains supported for compatibility. A bare UUID or identifier is resolved in the current project (or the project selected by `--project-id`) in this order: figure UUID, Organizer item UUID, figure identifier, then idea space drawing identifier. It is never interpreted as a project UUID. Use `--project-id` to identify an open project explicitly and `--id` to make an item target explicit.

If Curio is not already running, a `curio://` URL or `.curio` project path launches it. The CLI then waits for Curio's automation service and confirms the navigation rather than reporting success merely because LaunchServices accepted the URL. Item UUIDs, identifiers, and project UUIDs require the app to be running because they resolve against open projects.

Project targets must use an absolute path; `~/...` paths are expanded. Bare filenames and relative paths such as `./MyProject.curio` are rejected because multiple projects may share the same filename. Navigating to a saved project path opens it if necessary and makes an already-open project current. If Curio is hidden, it is unhidden and the destination is reasserted after the window transition. Save an untitled project first so it has a stable path that scripts can navigate to.

On success, `result` confirms the destination with `target_kind`, `project_uuid`, `path`, and `title`. It also includes `section_id`, `organizer_item_id`, or `figure_id` when those targets resolve. Therefore `response_ok: true` means Curio completed and verified the requested navigation. A missing project, unresolved item, or unconfirmed activation returns `response_ok: false`.

Available Fields

Field names are always written in `snake_case` and are passed as a comma-separated list to `--fields`.

When `--fields` is omitted, `query` and `get` return only the compact default fields `id`, `kind`, and `title`. This default is the same whether the command uses the running app or direct `--project` mode. Scripts should name every field they depend on explicitly; use `--fields all` to return every field that applies to each result item's type.

For example:

```
curio query "#active" --fields id,title,priority,due_date
```

The exact field list for your installed version is available via `curio fields`. For convenience, the full reference is split below into:

- Figure Fields
- Organizer Item Fields

`curio fields` shows the combined catalog across both target types, and `curio fields <name>` shows the type and description for one field.

Available Tags

Use `curio tags` to list all visible tags in the current project.

Examples:

```
curio tags
curio tags --project ~/Documents/MyProject.curio
```

`curio tags` returns:

- `items` : array of tag name strings
- `count` : number of returned tags

Tag names are formatted exactly as Curio expects them in tag-oriented CLI operations:

- Named tag sets use `TagSet/Tag Name` , such as `GTD/Active` or `Context/Next Action` .
- Project-local keyword tags use just `Tag Name` .
- Global keyword tags use `/Tag Name` .

When you set `tags`, Curio matches tag names case-insensitively and ignores spaces and emoji, so inputs such as `vacation,gtd/onhold` still resolve to the corresponding visible tags.

Creating Tags and Tag Sets

The running app can idempotently create project or global tags. With no scope option, creation targets the active project:

```
# Project keyword tags in the active project
curio tags create --tags "homework, study guide, exam"

# Global keyword tags
curio tags create --global --tags "homework, study guide, exam"

# A project tag set and its tags
```

```
curio tags create --tagset "GTD" --tags "done, on hold, finished"

# A global tag set and its tags
curio tags create --global --tagset "GTD" --tags "done, on hold, finished"
```

Use `--one-choice` when creating or extending a named set that should allow only one associated tag at a time, such as a status set used to group a Kanban board:

```
curio tags create --tagset "Status" --one-choice --tags "todo, active, on hold, done"
```

`--tagset` can be used without `--tags` to create an empty named set. If the set already exists, Curio extends it. If `--one-choice` is omitted, an existing set keeps its current one-choice setting.

Creation is idempotent: matching ignores case, spaces, and emoji, and existing tags or tag sets are reused rather than duplicated. If the selected project or global scope already contains multiple tag sets with the same normalized name, Curio rejects the request instead of choosing one arbitrarily. The response separates `created` and `existing` names and reports the resolved scope and tag-set properties:

```
{
  "scope": "project",
  "created": ["Status/todo", "Status/active"],
  "existing": ["Status/done"],
  "tag_set": {
    "name": "Status",
    "created": true,
    "one_choice": true,
    "uuid": "..."
  }
}
```

As with the other live project-scoped commands, use `--project-id <projectUUID>` to target a particular open project without activating it. Project UUIDs are available from `curio status`. For tag creation, this option cannot be combined with `--global`.

`--project <path>` remains the offline, read-only listing mode and is intentionally not accepted by `tags create`.

Querying

The `query` command uses the same query language as Curio's Search shelf and Quick Find, so the same query expressions, scopes, grouping, and sorting rules apply. See Curio's query language documentation for a deeper discussion of query syntax.

When Curio is running, context scopes such as `scope:section` and `scope:ideaspace` use the current Organizer section or active idea space. Direct `--project` mode is offline and returns `InvalidRequest` for context-dependent scopes that require live UI state.

Unless the query text includes an explicit `group:` command, the CLI returns a flat `items` array and reports `groupBy` as `null`. If the query text includes `group:`, the response switches to a `groups` array with `label` and `items` for each group.

A query containing only shaping commands such as `sort:title limit:400` has no criterion and therefore matches nothing. Use `not kind=nothing` to match all searchable items before applying those commands:

```
curio query "not kind=nothing sort:title limit:400" --fields id,title
```

Examples:

```
curio query "ipad"
curio query "(ipad or iphone) @SteveJ #event #2010" --fields id,title
curio query "#active due soon sort:priority" --fields id,title,priority,due_date
curio query "#active group:priority" --fields id,title,priority
curio query --project ~/Documents/MyProject.curio "#active" --fields id,title
curio query --project ~/Documents/MyProject.curio "kind=ideaspace modified > -3m sort:-modified limit:40" --fields id,titl
```

The final example is suitable for building a visual gallery of recently modified idea spaces. `kind=ideaspace` selects Organizer idea spaces, `modified > -3m` limits the results to the past three months, `sort:-modified` puts the newest first, and `limit:40` caps the gallery. Each result includes its stored preview path and a `curio://` link that opens Curio at that idea space.

You can optionally truncate long text fields in the response:

```
curio query "#meeting" --fields id,title,plain_text --max-chars 500
```

Use `--fields all` with `query` to return every field that applies to each result item's type.

Reading Selected Targets

Use `get --selected <kind>` to read specific live UI targets without constructing a query. These selected targets only work when Curio is running.

`--selected` value	Returns
`figure`	The single selected figure
`figures`	All selected figures
`section`	The current section containing the active organizer item
`sections`	All selected sections
`organizer_item` / `ideaspace`	The active Organizer item or idea space
`organizer_items` / `ideaspaces`	The selected Organizer items or idea spaces

Examples:

```
curio get --selected figure --fields plain_text,priority
curio get --selected figures --fields id,title,priority
curio get --selected section --fields id,title,kind
curio get --selected sections --fields id,title,kind
curio get --selected ideaspace --fields id,title
curio get --selected ideaspaces --fields id,title,kind
```

Singular targets return one object directly in `result`. Plural targets return an `items` array and `count`.

> **Important**
> Selected targets are based on Curio's current UI state, so they are not available with `--project`.

Figure Fields

These are the fields available when the target is a figure. You can use them with `query`, `get`, and figure-targeted `set`.

Use `--id <uuid-or-identifier>` to target a specific figure. UUID lookup is tried first; identifiers must be unique or Curio returns `InvalidRequest`. Use `--selected figure` to target the single selected figure.

`get` supports both the running app and `--project`. Use `--fields all` with `get` to return the full field set for the selected target type. `set` requires the running app.

Examples:

```
curio get --selected figure --fields raw_text,due_date,tags
curio get --id 12345678-1234-1234-1234-123456789ABC --fields id,title,kind,link
curio get --project ~/Documents/MyProject.curio --id 12345678-1234-1234-1234-123456789ABC --fields title,plain_text
curio get --id 12345678-1234-1234-1234-123456789ABC --fields link,asset_id,asset_file_kind,display_page
curio set --selected figure --priority 4
curio set --selected figure --rating 5 --progress 75
curio set --selected figure --progress 60 --checkmark-visible true
curio set --selected figure --start-date 2026-03-20 --due-date 2026-03-27
curio set --selected figure --done-date 2026-03-27T14:30
curio set --selected figure --caption "Sprint planning whiteboard"
curio set --selected figure --url "x-devonthink-item://932493923"
curio set --selected figure --note "Reviewed with team"
curio set --selected figure --field priority=3 --field rating=4
curio set --selected figure --field size.width=360 --field origin.x=72 --field origin.y=96
curio set --selected figure --field text_font_size=18 --field text_color=#123456 --field text_bold=true
curio set --selected figure --field fill_color=#ffcc66 --field pen_color=#112233 --field pen_width=2
curio set --selected figure --field displaying_as=preview --field display_page=2
curio set --selected figure --field icon_size=64 --field caption_within_shape_border=false
curio set --selected figure --field freeform_sizing=false --field min_height=120
curio set --selected figure --values '{"bounds":{"x":72,"y":96,"width":360,"height":240}}'
curio set --selected figure --values '{"text_alignment":"center","text_vertical_alignment":"middle"}'
curio set --selected figure --values '{"fill_style":"solid","shape":"RoundedRect","opacity":0.75}'
curio set --selected figure --values '{"note":"Reviewed with team","priority":3}'
curio set --selected figure --values-file updates.json
cat notes.md | curio set --selected figure --raw-text --stdin
```

For common fields, use the named flags such as `--priority` or `--due-date`. For script-generated updates, `--field key=value` handles quick scalar assignments, while `--values` and `--values-file` accept a JSON object of writable field values. Set `size.width` to let Curio recompute a natural height for figures that support it, such as text figures, image previews, and collections. Text styling fields use the `text_` prefix, including `text_font_size`, `text_color`, `text_bold`, and alignment fields. Figure styling fields include `fill_color`, `pen_color`, `pen_width`, `opacity`, `corner_radius`, `fill_style`, `shape`, and `dashed_pattern`. Asset presentation fields include `displaying_as`, `display_page`, caption placement, icon size, automatic sizing, freeform sizing, and minimum height. Use `--url` to set an asset figure's associated URL. Values without a URI scheme default to `https://`, so `apple.com` becomes `https://apple.com`. Schemes such as `mailto:` and `x-devonthink-item:` are preserved. All set forms merge into the same update request, so don't provide the same field more than once in a single command.

Setting `raw_text` is treated as a committed text edit. Before the command returns, Curio writes file-backed text to its underlying asset file, exports to any writable sync file, and propagates the text to synced figure instances and special idea-space text figures.

Field	Settable	Description
`id`	no	Figure or Organizer item UUID (stable, used for addressing)
`identifier`	no	User-visible figure identifier set via Curio's Info panel
`kind`	no	Figure or Organizer item kind used by query `kind` filters
`title`	no	Processed plain text title (writable when targeting an Organizer item)
`link`	no	`curio://` deep link URL that opens the project and navigates to the target item
`section_path`	no	Absolute section path beginning with `/` and ending with the current or containing section
`organizer_path`	no	Absolute Organizer path within the containing section, ending with the Organizer item or figure's idea space
`parent_id`	no	Immediate Organizer parent UUID, or containing idea space UUID for a figure
`raw_text`	yes	Raw text content (may contain markdown, MathJax, Mermaid source)
`caption`	yes	Asset figure caption text (empty string clears it)
`displaying_as`	yes	Asset figure display mode: `icon`, `preview`, `text`, or `live_preview` when supported
`display_page`	yes	1-based displayed page number for PDF asset figures
`caption_within_shape_border`	yes	Whether the asset figure caption is drawn within the shape border
`icon_size`	yes	Asset figure icon size in points, from `16` to `512`; sets square icon width and height
`auto_sizing`	yes	Whether the asset figure automatically sizes itself to its content
`freeform_sizing`	yes	Whether the asset figure uses freeform sizing instead of automatic sizing or minimum-height fitting
`min_height`	yes	Asset figure minimum height in points; `0` clears it
`plain_text`	no	Processed searchable text
`rendered_text`	no	Fully processed plain text
`rendered_markdown`	no	Figure content converted to Markdown
`url`	yes	Asset figure's associated URL; values without a URI scheme default to `https://`, while custom schemes are preserved
`asset_file`	no	File path to the underlying asset file (PDF, image, doc, etc.)
`preview_image_file`	no	Absolute path to an existing stored JPEG or PDF preview; `null` when no preview file exists
`asset_id`	no	Underlying asset UUID for an asset-backed figure
`asset_file_kind`	no	Underlying asset display/file kind
`sync_file`	no	Sync file URL, if sync is configured
`sync_file_direction`	no	Sync direction: `bidirectional`, `export`, or `import`
`start_date`	yes	Start date (`YYYY-MM-DD` or `YYYY-MM-DDTHH:MM`)
`due_date`	yes	Due date (`YYYY-MM-DD` or `YYYY-MM-DDTHH:MM`)
`done_date`	yes	Done date (`YYYY-MM-DD` or `YYYY-MM-DDTHH:MM`)
`added_date`	no	Date the figure or Organizer item was added (`YYYY-MM-DD` or `YYYY-MM-DDTHH:MM`, read-only)
`modified_date`	no	Date the figure or Organizer item was last modified (`YYYY-MM-DD` or `YYYY-MM-DDTHH:MM`, read-only)
`tags`	yes	Array of tag names; matching ignores case, spaces, and emoji when setting, for example `vacation,gtd/onhold`

Field	Settable	Description
`resources`	no	Array of resource names
`references`	no	Outgoing references grouped by reference type name; each value is an array of target `curio://` links
`priority`	yes	Priority level (`0`=none, `1`=very low, `2`=low, `3`=medium, `4`=high, `5`=urgent)
`rating`	yes	Rating (`0-5`)
`progress`	yes	Percent complete (`0-100`)
`checkmark_visible`	yes	Whether the figure's progress checkbox is visible
`text_font_family`	yes	Text font family name
`text_font_face`	yes	Text font face name within the selected family, such as `Regular`, `Bold`, or `Italic`; PostScript font names are also accepted
`text_font_size`	yes	Text font size in points
`text_color`	yes	Text foreground color as `#rgb`, `#rgba`, `#rrggbb`, or `#rrggbbaa`
`text_background_color`	yes	Text background color as `#rgb`, `#rgba`, `#rrggbb`, or `#rrggbbaa`; transparent clears the background
`text_bold`	yes	Whether text is bold
`text_italic`	yes	Whether text is italic
`text_underline`	yes	Whether text is underlined
`text_strikethrough`	yes	Whether text has strikethrough
`text_alignment`	yes	Text horizontal alignment: `left`, `center`, `right`, `justified`, or `natural`
`text_vertical_alignment`	yes	Text vertical alignment: `top`, `middle`, or `bottom`
`origin.x`	yes	Figure bounds origin x coordinate
`origin.y`	yes	Figure bounds origin y coordinate
`size.width`	yes	Figure width; recomputes natural height when supported
`size.height`	yes	Figure height
`bounds`	yes	Rectangle object with `x`, `y`, `width`, and `height`
`rotation`	yes	Figure rotation angle in degrees
`fill_color`	yes	Figure fill color as `#rgb`, `#rgba`, `#rrggbb`, or `#rrggbbaa`; setting a figure fill color enables solid fill unless `fill_style` is also set to `none`
`pen_color`	yes	Figure pen color as `#rgb`, `#rgba`, `#rrggbb`, or `#rrggbbaa`; setting a pen color ensures a visible pen width unless `pen_width` is also set to `0`
`pen_width`	yes	Figure pen width in points; `0` hides the stroke
`opacity`	yes	Figure opacity from `0.0` transparent through `1.0` opaque
`corner_radius`	yes	Figure corner radius in points, from `0` to `30`, when the figure or collection layout supports corner radius changes
`fill_style`	yes	Figure fill style: `none` or `solid`
`shape`	yes	Figure shape name, such as `Rectangle`, `RoundedRect`, `Capsule`, `Circle`, `Diamond`, or `Cloud`
`dashed_pattern`	yes	Figure dash pattern as `pixelsOn{,pixelsOff,pixelsOn,pixelsOff}{;phase}`; use `0` for a solid line
`note`	yes	Meta note text

`checkmark_visible` and `progress` are independent. Showing or hiding the progress checkbox does not change the percentage, and setting a percentage does not make the checkbox visible. Set both fields when the figure should behave as a visible task.

For MCP, pass `checkmark_visible` as a boolean property to `curio_set`; it is also accepted inside the generic `values` object.

Asset Presentation Fields

Asset figures support display, caption, and sizing controls through generic writable fields. Curio rejects display modes that the target figure cannot support, such as `text` for a plain image asset or `live_preview` for a non-URL asset. For PDFs, `display_page` is 1-based and out-of-range errors include the valid page range when Curio can read it.

`freeform_sizing` and `min_height` are mutually exclusive in Curio's figure model: enabling freeform sizing clears `min_height` and disables `auto_sizing`; setting `min_height` above `0` turns freeform sizing off.

Organizer Item Fields

These are the fields available when the target is an Organizer item or idea space. You can use them with `query`, `get`, and `set`.

Use `--id <uuid-or-identifier>` to target a specific Organizer item or idea space. UUID lookup is tried first; identifiers must be unique or Curio returns `InvalidRequest`. Use `--selected ideaspace` to target the active organizer item / idea space. `organizer_item` is still accepted as an alias.

`get` supports both the running app and `--project`. Use `--fields all` with `get` to return the full field set for the selected target type. `set` requires the running app.

Examples:

```
curio get --id 87654321-4321-4321-4321-CBA987654321 --fields title,note,tags
curio get --selected ideaspace --fields title,note,tags
curio set --selected ideaspace --title "Deep Link"
curio set --selected ideaspace --tags "GTD/Active,Planning"
curio set --selected ideaspace --note "Reviewed with team"
curio set --selected ideaspace --field dimension.width=1600 --field dimension.height=1000
curio set --selected ideaspace --values '{"dimension":{"width":1600,"height":1000},"fill_color":"#f0f7ff"}'
```

Idea space and Organizer item IDs support direct `title`, `note`, and `tags` reads and writes. Idea spaces also support `dimension.width`, `dimension.height`, and `fill_color`; these change the canvas size and background fill, not figure bounds or figure fill.

Field	Settable	Description
`id`	no	Idea space or Organizer item UUID (stable, used for addressing, identical to the underlying asset UUID)
`identifier`	no	User-visible idea space identifier set via Curio's Info panel
`kind`	no	Figure or Organizer item kind used by query `kind` filters
`asset_file_kind`	no	Underlying asset display/file kind
`title`	yes	Processed plain text title (writable when targeting an Organizer item)
`link`	no	`curio://` deep link URL that opens the project and navigates to the target item
`preview_image_file`	no	Absolute path to an existing stored JPEG or PDF preview; `null` when no preview file exists
`section_path`	no	Absolute section path beginning with `/` and ending with the current or containing section
`organizer_path`	no	Absolute Organizer path within the containing section, ending with the Organizer item or figure's idea space
`parent_id`	no	Immediate Organizer parent UUID, or containing idea space UUID for a figure
`dimension.width`	yes	Idea-space canvas width in points
`dimension.height`	yes	Idea-space canvas height in points
`fill_color`	yes	Idea-space background fill color as `#rgb`, `#rgba`, `#rrggbb`, or `#rrggbbaa`
`added_date`	no	Date the figure or Organizer item was added (`YYYY-MM-DD` or `YYYY-MM-DDTHH:MM`, read-only)
`modified_date`	no	Date the figure or Organizer item was last modified (`YYYY-MM-DD` or `YYYY-MM-DDTHH:MM`, read-only)
`tags`	yes	Array of tag names; matching ignores case, spaces, and emoji when setting, for example `vacation,gtd/onhold`
`references`	no	Outgoing references grouped by reference type name; each value is an array of target `curio://` links
`note`	yes	Meta note text

`section_path` and `organizer_path` are separate absolute, title-based paths. Both begin with `/` and include their final component. For an Organizer item in a nested section, values might be `/Clients/Acme/Research` and `/Filings/Contracts/Executed`. A figure reports the same `section_path` as its containing idea space, while its `organizer_path` ends with that idea space. Single-section projects still report a value such as `/Default Section`.

Keep the two fields separate rather than concatenating them, because the boundary between the section hierarchy and Organizer hierarchy would be lost. Use UUIDs and `parent_id` when stable, unambiguous addressing is required.

Creating Organizer Structure

Use `create` to add sections, folders, and idea spaces through the running Curio app. Creation is app-only so Curio's Organizer model, validation, selection handling, and undo behavior stay in charge; it is not available with `--project`.

Examples:

```
curio create section --title "Correspondence" --note "Reviewed" --tags "Discovery/Verified"
curio create section --parent 12345678-1234-1234-1234-123456789ABC --title "Authorities"
curio create folder --section 12345678-1234-1234-1234-123456789ABC --title "Discovery"
curio create ideaspace --section 12345678-1234-1234-1234-123456789ABC --title "001 - Notice of Motion"
curio create ideaspace --selected ideaspace --position next_sibling --title "002 - Exhibit A"
```

`create section` can target the Organizer root or a parent section. `create folder` and `create ideaspace` can target a section or another Organizer item. Every successful create response returns the new item's stable `id`, `kind`, `title`, `parent_id`, resolved `index`, and any create-time `note` or `tags` so scripts can chain later operations.

Moving Figures And Idea Spaces

Use `move` through the running Curio app to reposition a figure on the current idea-space canvas or to reparent/reorder an Organizer item such as an idea space. `move` is app-only and is not available with `--project`.

Figure moves use an absolute idea-space coordinate:

```
curio move --id 12345678-1234-1234-1234-123456789ABC --position 100,100
curio move --selected figure --position 240,160
```

Organizer moves use the same placement vocabulary as `create`:

```
curio move --id 12345678-1234-1234-1234-123456789ABC --relative-to 87654321-4321-4321-4321-CBA987654321 --position previou
curio move --id 12345678-1234-1234-1234-123456789ABC --section 87654321-4321-4321-4321-CBA987654321 --position last_child
curio move --selected ideaspace --parent 87654321-4321-4321-4321-CBA987654321 --position first_child
```

Options:

- `--id <uuid>` targets a figure or Organizer item.
- `--selected figure|ideaspace|organizer_item` uses Curio's current UI selection.
- `--position x,y` moves a figure to an absolute canvas coordinate.
- `--position first_child|last_child|first_sibling|previous_sibling|next_sibling|last_sibling` moves an Organizer item.
- `--parent <uuid>`, `--section <uuid>`, and `--relative-to <uuid>` choose the Organizer destination context.
- `--index <n>` provides an explicit child index for Organizer moves and cannot be combined with `--position`.

Successful move responses include `mode` (`figure` or `organizer`) plus the moved item's `id`. Figure responses include final geometry fields such as `origin.x`, `origin.y`, and `bounds`; Organizer responses include final `parent_id` and `index`.

Command Batch Files

Use `curio batch` to run a UTF-8 text file containing one Curio command per line. Lines can omit the leading `curio` prefix, blank lines are ignored, and `#` starts a comment outside quotes.

```
# filing.curio-batch
discovery = create section --title "Discovery"
emails = create ideaspace --title "Emails" --parent $discovery.id
import "~/Documents/emails.pdf" --as spread-pdf --parent $discovery.id --layout layout.json
```

```
curio batch filing.curio-batch
```

A line can store its structured result with `name = command ...`. Later lines can refer to top-level result fields such as `$discovery.id`, `$emails.link`, or `$spread.asset_id`.

A reference is always substituted as a single argument, so values containing spaces need no quoting. A reference that resolves to a value beginning with `--` is rejected, because project text such as a figure title would otherwise be read as a command option.

Batch output is JSONL: each executed command writes one compact JSON object on its own line. There is no enclosing JSON document or top-level `response_ok` value for the complete batch. Each line object contains `line`, `command`, `ok`, `exit_code`, `ref` (`null` for unassigned lines), and either `result` or `error`:

```
{"command":"create","exit_code":0,"line":2,"ok":true,"ref":"discovery","result":{"id":"..."}}
{"command":"save","exit_code":0,"line":3,"ok":true,"ref":null,"result":{"saved":true}}
```

Scripts should read stdout one line at a time. Execution stops on the first failure unless you pass `--continue-on-error`, so the final emitted object may describe the failed line.

Batch files do not support child commands that read from stdin, such as `import --stdin` or `set --stdin`, because the batch file itself owns the input stream. Use file-backed imports or pass values directly in the batch line.

Placement options:

Position	Meaning
`first_child`	Insert as the first child of the target parent.
`last_child`	Insert as the last child of the target parent. This is the default.
`first_sibling`	Insert at the beginning of the reference item's sibling list.
`previous_sibling`	Insert immediately before the reference item.
`next_sibling`	Insert immediately after the reference item.
`last_sibling`	Insert at the end of the reference item's sibling list.

Target options:

- `--parent <uuid>` targets a parent section or Organizer item.
- `--section <uuid>` targets a section for folder or idea-space creation.
- `--relative-to <uuid>` targets an existing section or Organizer item for sibling placement.
- `--selected section|ideaspace|organizer_item` uses Curio's current UI selection.
- `--index <n>` provides an explicit child index and cannot be combined with `--position` .

Sections

Use `sections` to return the Organizer's section hierarchy. This command returns a recursive tree by default using `children`, so child sections are nested under their parent sections automatically.

Examples:

```
curio sections
curio sections --project ~/Documents/MyProject.curio
```

Top-level special sections such as Archive, Trash, and Journal are included when present.

Sections Result

`curio sections` returns structured JSON directly and does not use `--fields`:

- `items` : array of top-level section objects
- `count` : number of top-level section objects

Each section object includes the same general metadata style as other CLI output, and parent sections include a `children` array when they contain child sections.

Assets

Use `assets` to list assets in the project or within a specific section.

Examples:

```
curio assets
curio assets --project ~/Documents/MyProject.curio
curio assets --project ~/Documents/MyProject.curio --section 12345678-1234-1234-1234-123456789ABC
curio assets --project ~/Documents/MyProject.curio --parent 87654321-4321-4321-4321-CBA987654321
curio assets --project ~/Documents/MyProject.curio --section 12345678-1234-1234-1234-123456789ABC --recursive --organizer_
curio assets --project ~/Documents/MyProject.curio --parent 87654321-4321-4321-4321-CBA987654321 --recursive --organizer_o
```

`assets --section <uuid>` restricts results to a section. `assets --parent <uuid>` restricts results to an Organizer parent such as a folder, idea space, or section. Both are non-recursive by default. Add `--recursive` to include child Organizer items within that same section.

Sections are independent project divisions. Recursion follows folders and Organizer items but does not cross into child sections. To process a complete project, walk the hierarchy returned by `curio sections`, including nested `children`, and issue a separate `assets --section <uuid>` request for each section.

The returned asset list includes figure assets that are contained within idea spaces, like PDF images. File-backed assets include `asset_file`, so a normal flat `curio assets` response can serve as a bulk file manifest without one `get` call per figure. Organizer-only listings produced by `--organizer_order` generally contain idea spaces and folders rather than their underlying figure assets, so use normal assets mode when collecting file paths.

Asset results also include `link` when a `curio://` destination is available. Idea spaces or asset-backed figures with an existing stored preview include `preview_image_file`. Stored previews may be JPEG or PDF. Reading them is non-mutating: the CLI does not generate or refresh missing previews.

Assets Result

`curio assets` returns structured JSON directly and does not use `--fields`:

- `items` : array of asset objects
- `count` : number of returned assets
- `total_count` : in Organizer-order mode, number of top-level and nested Organizer items

In normal mode, `items` is a flat array. Use `--section <uuid>` to start from a section or `--parent <uuid>` to start from any Organizer parent. With `--organizer_order`, `items` follows the Organizer order and organizer folders include nested `children`; `count` remains the size of that top-level `items` array and `total_count` includes the nested descendants. Combine `--recursive` with `--organizer_order` to include descendants in Organizer order within the requested section. Section recursion is explicit: walk `curio sections` and request each section UUID separately. This Organizer-order mode only returns items that appear in the Organizer UI for that parent.

The hierarchy fields `parent_id`, `parent_title`, `depth`, `index`, and `container_kind` are included only when `--organizer_order` is specified.

Each asset object may include:

Field	Type	Description
`id`	string	Asset UUID
`type`	string	Asset object type
`kind`	string	Stable Curio asset kind for scripting, such as `organizerfolder`, `organizersection`, or `image`
`asset_file_kind`	string	Human-readable asset display/file kind, if applicable
`title`	string	Asset title
`tags`	`[string]`	Asset tags
`note`	string	Asset note text, if present
`asset_file`	string	Underlying file path, if the asset has one
`preview_image_file`	string	Existing stored JPEG or PDF preview path, if the asset has one
`link`	string	`curio://` link for an Organizer item, if applicable
`section_path`	string	Absolute path ending with the current or containing section
`organizer_path`	string	Separate absolute Organizer path within the containing section, including the item itself when applicable
`parent_id`	string	UUID of the containing Organizer parent; only present with `--organizer_order`
`parent_title`	string	Title of the containing Organizer parent; only present with `--organizer_order`
`depth`	number	Hierarchy depth relative to the requested section or parent; only present with `--organizer_order`
`index`	number	Zero-based position within the containing Organizer parent; only present with `--organizer_order`
`container_kind`	string	Kind of Organizer parent containing the asset, such as `section`, `folder`, or `ideaspace`; only present with `--organizer_order`
`added_date`	string	Added date in `YYYY-MM-DD` or `YYYY-MM-DDTHH:MM` format
`modified_date`	string	Modified date in `YYYY-MM-DD` or `YYYY-MM-DDTHH:MM` format

Organizer contents

Technically Organizer items are assets so you can use `curio assets` to get your Organizer tree as well.

- Use `--organizer_order` to return only organizer-visible items in the exact Organizer UI order for that section or parent, with folder contents nested under `children` .
- `--organizer_order` requires `--section` or `--parent` and can be combined with `--recursive` .
- In Organizer-order mode, `count` is the number of top-level objects in `items`, while `total_count` includes all descendants nested under `children`.
- When `--organizer_order` is specified, results include `parent_id` , `parent_title` , `depth` , `index` , and `container_kind` so callers can reconstruct the Organizer tree. Flat asset listings do not include these hierarchy fields.
- This mode is Organizer-facing, so it does not include figure assets that are contained within idea spaces, like PDF images.

To flatten or count the complete Organizer tree with `jq`:

```
curio assets --section <uuid> --recursive --organizer_order |
  jq '[.result.items[] | recurse(.children[]?)] | length'
```

Example Organizer tree:

```
Case 25
├── Deep Link
└── Deep Folder
    └── Item in Folder
```

The UUID for each returned asset can be used as the idea space / Organizer item UUID in calls such as `set` and `get`.

Each asset includes both a machine-readable `kind` and a human-readable `asset_file_kind`:

- `kind` is the stable Curio asset kind for scripting and matches the query language's `kind=...` filter, with values such as `organizerfolder`, `organizersection`, `organizerideaspace`, `organizerspecialsection`, or `image`.
- `asset_file_kind` is the asset's display/file kind, such as `Organizer Folder` or `PDF document`.

This split lets scripts filter by Curio structure using `kind` while still inspecting file-specific details using `asset_file_kind` and `asset_file`.

Asset files, previews, and sidecars

The `asset_file` path points to the file stored by or linked to the Curio asset. For an embedded asset, scripts may place related sidecar files in the asset's containing folder. Curio keeps the whole folder together through asset renames and Organizer reorganization. If the asset is deleted, Curio deletes its containing folder, including any sidecars.

The read-only `preview_image_file` field points to an existing stored preview for an idea space or asset-backed figure. An explicitly requested field is `null` when no preview file exists; bulk `assets` results omit the field in that case. Preview files may be JPEG or PDF. Reading the field does not generate or refresh a missing or stale preview.

For shared Spread PDF assets, replacing `asset_file` while Curio is closed preserves the existing figure UUIDs and `display_page` references. Use a replacement with the same page count. Curio does not key ordinary PDF assets by content hash; sync-file hashing applies only to `sync_file` workflows.

Exporting

The `export` command exports content from the running app to a file or the clipboard. You can export a single figure by UUID or identifier, all selected figures, or selected idea spaces in the Organizer.

Supported export formats are `png`, `jpg`, `pdf`, `text`, `markdown`, and `rtf`.

Examples:

```
curio export --id 12345678-1234-1234-1234-123456789ABC --format png --filename ~/Desktop/Figure
curio export --id 87654321-4321-4321-4321-CBA987654321 --format pdf --filename ~/Desktop/IdeaSpace
curio export --selected figures --format png --filename ~/Desktop/Selection
curio export --selected figures --format pdf --clipboard
curio export --selected figures --format markdown --filename ~/Desktop/Notes --include-assets
curio export --selected ideaspaces --format pdf --filename ~/Desktop/IdeaSpaces
curio export --selected ideaspace --format pdf --clipboard
curio export --selected figures --format png --filename ~/Desktop/Selection --overwrite
```

If `--filename` points to an existing folder then Curio will automatically name the export using the selected content's title.

Without `--overwrite` an existing destination is never replaced. Even with `--overwrite`, exports refuse to replace your home folder and its standard subfolders, hidden configuration items in your home folder such as `~/.zshrc` and `~/.ssh`, and the `LaunchAgents`, `LaunchDaemons`, and `StartupItems` folders.

If `--id` names a figure, Curio navigates to that figure and exports the resulting single selection. If `--id` names an idea space, Curio exports that idea space directly.

> **Note**
> `--clipboard` is supported for figure exports, and for `--selected ideaspace` only when exactly one idea space is selected and the format is `png`, `jpg`, or `pdf`. For file exports, `--selected ideaspace` supports `png` and `jpg` only when exactly one idea space is selected. Multi-idea-space file exports are supported for `pdf`, `text`, `markdown`, and rich text.

> **Note**
> Rich text exports are requested with `--format rtf`, but the actual output may be `.rtf` or `.rtfd` depending on whether attachments are present. Markdown exports with `--include-assets` use a sibling `Assets/` folder, and repeated exports merge into that folder instead of replacing it. Exported markdown asset references are written to match the resulting asset folder layout.

Importing

The `import` command creates a new figure on the current idea space from a file or from standard input. It can also spread a multi-page PDF into one idea space per page. It requires the running Curio app and does not support `--project`.

Important
The Mac App Store edition of Curio runs in a sandbox that limits which file paths it can access. File import (`--as file`), Spread PDF import (`--as spread-pdf`), and file sync (`--sync`) work only when the source file is in an **accessible folder** — the app's temporary directory, Downloads, your Curio projects folder, or any folder you have previously authorized in Curio. Run `curio status` to see the list of accessible folders for your installation. Content import (`--as text`, `list`, `mindmap`, `stack`, `table`) without `--sync` works with both editions regardless of file location, because the CLI reads the file and sends the content directly.

Tip
If you are scripting file imports against the Mac App Store edition, use the accessible folders list from `curio status` to find a staging location. The app's temporary directory is always available and can be written to by any process running as the same user. Note that the sandboxed temporary directory is inside the app container (e.g. `~/Library/Containers/com.zengobi.curio/Data/tmp/`), not the system `/tmp`. The website (direct download) edition is not sandboxed and can access any file path.

Importing a file as an asset figure

By default, `import` copies the file into the project as an asset figure (image, PDF, document, etc.):

```
curio import ~/Photos/diagram.png
curio import ~/Documents/report.pdf --transfer move
curio import /Volumes/Ext/video.mp4 --transfer alias
```

The `--transfer` flag controls how the file is handled:

Value	Description
`copy`	Copy into the project (default)
`move`	Move into the project
`alias`	Reference the file externally without copying

Spreading a PDF across idea spaces

Use `--as spread-pdf` to import a multi-page PDF as one stored PDF asset and one or more idea spaces. By default, Curio creates one idea space per requested page. Each created idea space contains a distinct PDF figure instance, so each page has its own figure UUID and `curio://` link while all figures share the same `asset_id` and `asset_file`.

Curio text and Markdown export describes the canvas; it does not extract text from embedded asset files. Scripts that need a PDF text layer or OCR output should process the returned `asset_file` externally and use each figure's `display_page` to associate page text with its Curio figure and deep link.

Preview regeneration when opening a project

- By default, Curio silently regenerates missing or stale idea space previews after a project opens. It generates one preview at a time and returns to the event loop between idea spaces, so you can continue navigating and editing without a progress window interrupting your work.
- By default, if an individual preview takes one second or longer and more previews remain, Curio pauses quiet regeneration for the rest of that project session to avoid a series of stalls. The completed preview remains fresh; outstanding previews can be updated manually or picked up the next time the project opens. You can fine-tune these thresholds and diagnostics.
- Incoming CLI automation takes priority. Curio stops preview generation after the current idea space, lets the CLI request run, then schedules another quiet preview pass after automation becomes idle.
- Closing the project or quitting Curio also stops the quiet pass after the current idea space. Any remaining stale previews are picked up the next time the project opens. Automatic PDF Mirror projects still regenerate required previews when publishing during close or quit.
- To maximize CLI speed by disabling this background work, click here to set `Preview Update On Project Open` to `no`. You can also use **Curio > Change Advanced Setting**. Set it back to `yes` to restore the default behavior.

```
curio import ~/Documents/affidavit.pdf --as spread-pdf --section <sectionUUID> --title-template "{page:03}"
curio import ~/Documents/affidavit.pdf --as spread-pdf --section <sectionUUID> --pages 1-10,15
curio import ~/Documents/affidavit.pdf --as spread-pdf --selected ideaspace --position next_sibling
curio import ~/Documents/affidavit.pdf --as spread-pdf --section <sectionUUID> --page-titles titles.json
curio import ~/Documents/affidavit.pdf --as spread-pdf --section <sectionUUID> --layout layout.json
curio import ~/Documents/affidavit.pdf --as spread-pdf --section <sectionUUID> --dry-run
```

Spread PDF placement uses the same target and position model as `curio create`:

Option	Description
`--section <uuid>`	Create page idea spaces under a section
`--parent <uuid>`	Create page idea spaces under a section or Organizer item
`--selected ideaspace`	Use the selected or active Organizer item as the placement reference
`--selected organizer_item`	Alias for `--selected ideaspace`
`--relative-to <uuid>`	Use an Organizer item UUID as the placement reference
`--position <pos>`	`first_child`, `last_child`, `first_sibling`, `previous_sibling`, `next_sibling`, or `last_sibling`
`--index <n>`	Numeric child index; cannot be combined with `--position`

By default, `--pages all` creates one idea space for every PDF page. Use a comma-separated page/range list such as `1-10,15` to import a subset. Duplicate pages are rejected.

Use `--title-template` for generated idea space titles. Supported tokens are `{page}`, `{page:03}`, `{page_start}`, `{page_end}`, `{page_count}`, and `{filename}`. Use `--page-titles` when titles need to come from a manifest:

```
[
  { "page": 1, "title": "2026-05-01 – Letter from Smith" },
  { "page": 2, "title": "p2" }
]
```

Use `--layout` when each idea space should contain a grid of PDF pages instead of a single page. The number of pages per idea space is `columns * rows`. The optional `figure` frame defines the first page figure, and `x_gap` / `y_gap` determine the offsets for the remaining columns and rows:

```
{
  "columns": 2,
  "rows": 2,
  "x_gap": 50,
  "y_gap": 50,
  "figure": { "x": 72, "y": 72, "width": 420, "height": 540 },
  "title_template": "Pages {page_start}–{page_end}"
}
```

A layout file can also include `pages`. Explicit command-line options such as `--pages` and `--title-template` override values in the layout file.

`--transfer copy` and `--transfer move` are supported. `--transfer alias` is rejected for Spread PDF import because the operation creates a stored project asset that is shared by the created page figures.

The response includes `asset_id`, `asset_file`, and an `items` array with every created `ideaspace_id`, title, page range, and page figure. For layout imports, each item includes a `figures` array with each figure's `figure_id`, `display_page`, frame, and link.

Importing content as a structured figure

Use `--as` to choose a specific import mode instead of the default asset-figure import:

```
curio import ~/notes.md --as text
curio import ~/outline.md --as list
curio import ~/outline.md --as mindmap
curio import ~/outline.md --as stack
curio import ~/data.csv --as table
```

`--as` value	Description
`file`	Store the file as an asset figure (default when `--as` is omitted)
`spread-pdf`	Store one PDF asset and create one idea space per requested page
`text`	Create a text figure from the file content
`list`	Parse hierarchical text into a list collection
`mindmap`	Parse into a mind map collection
`stack`	Parse into a stack collection
`table`	Parse tabular text (CSV/TSV/markdown table) into a table collection

The content format is auto-detected from the file extension:

Extension	Format
`.md`, `.markdown`	Markdown
`.taskpaper`	TaskPaper
`.opml`	OPML
`.csv`	CSV
`.tsv`	TSV
`.txt`, other	Plain text

Use `--ignore-inline-tags` or `--ignore-inline-resources` with a collection import (`list`, `mindmap`, `stack`, or `table`) to preserve inline `#tag` and `@resource` tokens in item text instead of converting them into hidden metadata. An ignored inline token does not create or associate a new tag or resource. Function-style metadata such as `@tags(...)` and `@resources(...)` is still processed normally.

```
curio import ~/outline.md --as list --ignore-inline-tags --ignore-inline-resources
```

Reading content from standard input

Use `--stdin` to read content from standard input instead of a file. This only works with content types (`--as text`, `list`, `mindmap`, `stack`, or `table`).

```
echo "- Item 1\n- Item 2" | curio import --stdin --as list
pbpaste | curio import --stdin --as text
cat data.tsv | curio import --stdin --as table --content-format tsv
```

Since stdin has no file extension, the content format defaults to Markdown (or CSV when `--as table`). Use `--content-format` to override this default:

```
curio import --stdin --as list --content-format taskpaper
curio import --stdin --as table --content-format tsv
curio import --stdin --as mindmap --content-format opml
```

Supported `--content-format` values: `markdown`, `taskpaper`, `opml`, `csv`, `tsv`, `plain`.

Setting up a sync file

Use `--sync` to create a content figure that stays synchronized with the source file. This requires a file path (not `--stdin`) and a content type (`--as text`, `list`, `mindmap`, `stack`, or `table`).

```
curio import ~/todo.md --as list --sync bidirectional
curio import ~/brainstorm.md --as mindmap --sync import
curio import ~/notes.md --as text --sync export
```

`--sync` value	Description
`bidirectional`	Changes flow both ways between the figure and the file
`export`	Figure changes export to the file
`import`	File changes import into the figure

For synced collection figures, Curio's Markdown exporter appends an HTML comment such as `<!--id:Ab3-->` to each item. Preserve those markers when an external generator rewrites the file: they allow Curio to retain each item's

figure UUID. Without them, imported entries are treated as new figures and can lose figure-only styling, metadata, or links.

Import response

The response includes the created figure's UUID so you can immediately use `get` or `set` on it:

```
{
  "api_version": 1,
  "app_version": "26.1",
  "response_ok": true,
  "result": {
    "figure_id": "12345678-1234-1234-1234-123456789ABC",
    "figure_kind": "list",
    "title": "todo"
  }
}
```

When `--sync` is used, the response also includes `sync_file` and `sync_direction`.

Ordinary file and content imports update the live project immediately, so the returned UUID can be used by `get`, `set`, or `query` without an intervening save. Before a subsequent read or save, Curio settles deferred figure synchronization, query-backed collection refreshes, cache invalidation, and document edited-state propagation. Imports do not force a disk save per item. Batch imports and related mutations, then call `curio save` once. Spread PDF imports are the exception: they settle and save once after the full spread is created.

Saving

Use `save` as a synchronous persistence barrier after a sequence of live mutations:

```
curio save
curio save --project-id 12345678-1234-1234-1234-123456789ABC
```

Without `--project-id`, Curio saves the current project. Use a `project_uuid` returned by `curio status` to save a different open project without bringing its window to the front. Before saving, Curio settles deferred model propagation; the command returns only after Curio has prepared every window controller and saved the project's Organizer state and page drawings.

An untitled project must first be saved through Curio so it has a stable path. Read-only projects cannot be saved through the CLI.

`curio save` does not close the project. If a script closes Curio through AppleScript, use `saving yes`; a bare `close` does not provide the same explicit persistence barrier. AppleScript's `documents` collection also contains Curio's internal Stencils and IdeaSpaceTemplates documents, whereas `curio status` lists user projects, so do not assume those collections have matching names or members.

JSON Output and Exit Codes

All commands print JSON responses. On success the response contains `response_ok: true`; on failure it contains `response_ok: false` plus an `error` dictionary.

A typical success response looks like this:

```
{
  "api_version": 1,
  "app_version": "25.1",
  "response_ok": true,
  "result": {
    "items": [
      {
        "id": "fig-abc",
        "title": "Write docs",
        "priority": 4
      }
    ]
  }
}
```

Exit codes are:

- `0` for success
- `1` for an API or runtime error
- `2` for invalid command line usage

Common Errors

Error	Meaning
`NoSelection`	Nothing is selected
`MultipleSelection`	More than one figure is selected when a single figure is required
`SelectionNotTextCapable`	The selected figure does not support text for the requested operation
`FigureNotFound`	The requested UUID or identifier could not be found
`ReadOnlyField`	You tried to write a read-only field
`ReadOnlyTransport`	You tried to write through a read-only transport
`NotActivated`	Curio needs to be launched to refresh CLI activation
`InvalidRequest`	The command or supplied values were not valid, or an identifier matched multiple targets
`Busy`	Another automation request is still finishing; the client retries briefly before returning this error
`InternalError`	An unexpected internal failure occurred

Tip
If you are scripting against the CLI, check the `response_ok` field in the JSON response and the process exit code.

Troubleshooting

If a command says it cannot connect, Curio is not running and the command requires the live app. Launch Curio and try again.

If `--project` mode returns `NotActivated`, launch Curio once so it can refresh the CLI activation token.

If a selection-based command fails, confirm that the correct project window is frontmost and that the expected figures or Organizer items are selected.

Settings

The CLI supports user settings in `~/.curio/settings.json`. Use `curio settings` to inspect and update the file instead of editing it by hand:

```
curio settings list
curio settings get query.max_chars
curio settings set query.max_chars=4000
curio settings set output.pretty_json=no
curio settings remove query.max_chars
curio settings path
curio settings reveal
```

Settings are addressed by dotted keys and stored as nested JSON. Explicit command-line flags still win over settings. Boolean settings accept `true` / `false`, `yes` / `no`, `on` / `off`, and `1` / `0`. Use `curio settings reveal` to create the settings file if needed and reveal it in Finder.

Setting	Type	Default	Description
`query.max_chars`	integer	unset	Default text truncation for `curio query`; explicit `--max-chars` wins.
`output.pretty_json`	boolean	`true`	Pretty-print normal JSON responses. Set to `no` for compact script output.
`output.color`	string	`auto`	Terminal styling: `auto`, `always`, or `never`.
`diagnostics.verbose`	boolean	`false`	Enable verbose diagnostics by default.
`batch.continue_on_error`	boolean	`false`	Continue batch execution after a failed line unless stopped by another fatal condition.
`mcp.log_path`	string	unset	Default log path for `curio mcp-server`; explicit `--log` wins.
`transport.send_timeout_seconds`	number	`10.0`	Timeout for sending requests to the running Curio app.
`transport.receive_timeout_seconds`	number	`10.0`	Timeout while waiting for the running Curio app to reply.
`curio.install_preference`	string	`auto`	Preferred Curio install for direct project reads when both editions are present: `auto`, `direct`, or `mas`.

AI Agents

The Curio CLI lets AI agents search, inspect, organize, and modify Curio projects. Agents that can run command-line tools work most efficiently with Curio's bundled skills. Other AI clients can connect through the CLI's Model Context Protocol (MCP) server.

> **Agents, assistants, and chatbots**
> People often call products such as ChatGPT and Claude *chatbots* or *AI assistants* because conversation is their primary interface. In this guide, *AI agent* is a capability-based term for an AI client that can use tools—such as Curio's CLI or MCP server—to take actions on your behalf. This includes coding agents such as Codex and Claude Code as well as more autonomous or persistent agent systems; it does not imply that the client runs unattended.

> **Before you begin**
> Install and verify the CLI as described in Installing the CLI. See Using the CLI for its commands, fields, transport modes, output, and errors.

Choosing an Integration

AI client	Recommended integration
ChatGPT desktop Work or Codex mode	Curio skills in `~/.agents/skills/` or `~/.codex/skills/`
Codex CLI	Curio skills in `~/.agents/skills/` or `~/.codex/skills/`
Claude Code CLI or Claude Desktop Code tab	Curio skills in `~/.claude/skills/`
Claude Desktop Chat or Cowork tab	Packaged Curio Claude Connector
Antigravity CLI or Desktop	Curio skills in the corresponding platform folder
ChatGPT web	Secure MCP Tunnel to the local Curio MCP server
Other MCP hosts	Curio's local stdio MCP server

Skills let a CLI-capable agent call `curio` directly. MCP hosts instead launch `curio mcp-server` and interact with its tools, prompts, and resources over standard input and output.

AI Agent Skills

The Curio CLI bundles skill files that teach CLI-capable AI agents how to use the `curio` command effectively. Installed skills are symlinks to files under `/Library/Application Support/CurioCLI/skills/`, so they automatically reflect the latest installed CLI version.

Two skills are installed by `curio skills install ...`:

- **curio** — all commands, transport modes, fields, workflows, and error handling.
- **curio-query** — query language syntax, including boolean logic, tags, dates, comparisons, sorting, grouping, scopes, and limits.

A third bundled skill, **curio-mcp**, remains available as reference material for legacy or manual hosts that support uploaded skill packages. Claude Desktop users should normally install the MCPB connector with `curio claude install` instead of uploading `curio-mcp-upload.zip`.

Codex and Claude Code can choose a skill automatically when a request matches its description, or you can invoke one explicitly. In Codex, mention `$curio` or `$curio-query`, or type `/skills` and choose it from the Skills UI. In Claude Code, use `/curio` or `/curio-query`.

See OpenAI's Codex skills documentation and Anthropic's Claude Code skills documentation for platform-specific details.

Listing Platforms

Use `curio skills` or `curio skills list` to see available platforms and their current installation state:

```
curio skills list
```

Each platform is reported as **installed**, **partial**, or **not installed**. Status is determined from the platform's current skill symlinks rather than a separate manifest. Platforms are detected from their configuration directories, such as `~/.agents/` or `~/.claude/`. A partial installation identifies the missing skills.

Installing Skills

Install the skills in the common folder used by most AI agents, including ChatGPT/Codex and Gemini/Antigravity but not Claude:

```
curio skills install agents
```

Or install them for a specific platform:

```
curio skills install claude
curio skills install codex
curio skills install agy
curio skills install antigravity
```

To detect installed platforms and install the skills for all of them:

```
curio skills install --all
```

For example, `curio skills install agents` creates:

- `~/.agents/skills/curio` → `/Library/Application Support/CurioCLI/skills/curio`
- `~/.agents/skills/curio-query` → `/Library/Application Support/CurioCLI/skills/curio-query`

The command does not install the bundled `curio-mcp` reference skill as a platform symlink. Installation is idempotent, so running the command again is a safe no-op. If a file or directory not managed by the CLI already exists at a destination, the installer skips it and prints a warning rather than overwriting it.

Uninstalling Skills

Remove skills for one platform:

```
curio skills uninstall claude
```

Or remove every Curio CLI-managed installation:

```
curio skills uninstall --all
```

Uninstall only removes symlinks that point to the CLI's bundled skills directory. It does not remove files or symlinks created manually or by another tool.

Browsing Bundled Skills

Open the bundled skills folder in Finder for inspection or manual copying to an unsupported platform:

```
curio skills reveal
```

Supported Platforms

Platform	Skill directory
Agents (common folder used by most AI agents)	`~/.agents/skills/`
Claude Code	`~/.claude/skills/`
Codex	`~/.codex/skills/`
Antigravity CLI (`agy`)	`~/.gemini/skills/`
Antigravity Desktop	`~/.antigravity/skills/`

Platforms that support manual skill copying can use `curio skills reveal`. The ChatGPT desktop app can use skills installed in either `~/.agents/skills/` or `~/.codex/skills/`.

Skills Command Summary

```
curio skills
curio skills install <platform>
curio skills install --all
curio skills uninstall <platform>
curio skills uninstall --all
curio skills reveal
```

MCP Server

The same `curio` binary can run as a stdio MCP server:

```
curio mcp-server
curio mcp-server --log ~/Library/Logs/curio-mcp.log
```

This mode is launched by an MCP host such as Claude Desktop. It is not a daemon or background service. The host starts the server on demand, communicates with it over stdin/stdout for the session, and then shuts it down.

The MCP server shares its command runner with the CLI so the two surfaces stay aligned.

MCP tool	CLI equivalent	Purpose
`curio_query`	`curio query`	Search figures using a Curio query expression.
`curio_get`	`curio get`	Read properties for a figure, idea space, Organizer item, or live selected target.
`curio_set`	`curio set`	Update writable properties.
`curio_create`	`curio create`	Create sections, folders, and idea spaces.
`curio_project_create`	`curio project create`	Create and open an empty project with a Default Section; a missing `.curio` extension is added.
`curio_move`	`curio move`	Move a figure or Organizer item.
`curio_batch`	`curio batch`	Run command batches with pseudo-variable references.
`curio_fields`	`curio fields`	List automation fields.
`curio_field_detail`	`curio fields <name>`	Describe one field and its usage.
`curio_docs`	bundled MCP docs	Read bundled MCP guidance.
`curio_tags`	`curio tags`	List project tags.
`curio_tags_create`	`curio tags create`	Create project/global keyword tags or named tag sets.
`curio_export`	`curio export`	Export a figure or live selection.
`curio_import`	`curio import`	Import files, content, or Spread PDF pages.
`curio_navigate`	`curio navigate`	Navigate to a project, section, idea space, or figure.
`curio_save`	`curio save`	Save the current or specified open project.
`curio_sections`	`curio sections`	List the section hierarchy.
`curio_assets`	`curio assets`	List figures and assets.
`curio_project_center`	`curio project-center`	List Project Center categories and project paths.
`curio_status`	`curio status`	Check app state, projects, transport, and selection availability.
`curio_help`	`curio help`	Show CLI help.
`curio_version`	`curio --version`	Show CLI version information.
`curio_update`	`curio update`	Check CLI update status.

`curio_docs` supports the topics `overview`, `query-language`, `workflows`, and `transport-modes`. It provides read-only guidance for hosts that do not expose MCP resources directly.

`curio_project_center` uses Curio's live Project Center when the app is available and otherwise reads the persisted Project Center and Recently Opened snapshots. Its `operation` is `categories` or `projects`; `projects` accepts an optional category UUID or title. Check `result.data_source` for `live` or `persisted`.

The server also exposes MCP-native resources at identifiers such as:

- `curio-mcp://docs/overview`
- `curio-mcp://docs/query-language`
- `curio-mcp://docs/workflows`
- `curio-mcp://docs/transport-modes`

It exposes prompts for workflows including `build_curio_query`, `inspect_curio_selection`, and `update_curio_item`.

MCP resource identifiers
`curio-mcp://docs/...` identifiers belong to the MCP server. An MCP client passes them back to the server during `resources/read`; macOS and Curio do not open them. They are intentionally separate from Curio's real `curio://` link scheme.

Manual MCP Host Configuration

Claude Desktop users should normally install the packaged connector. For other MCP hosts, use this stdio server configuration:

```
{
  "mcpServers": {
    "curio": {
      "command": "/Library/Application Support/CurioCLI/bin/curio",
      "args": ["mcp-server"]
    }
  }
}
```

If your host expects only the server entry, use:

```
{
  "command": "/Library/Application Support/CurioCLI/bin/curio",
  "args": ["mcp-server"]
}
```

Use `/Library/Application Support/CurioCLI/bin/curio` for host-launched MCP connections. This installed path remains stable across CLI updates.

Integrating with AI Clients

Shared Agents Folder

Many AI agent tools, including ChatGPT/Codex and Gemini/Antigravity but not Claude, use the common `~/.agents/` folder:

```
curio skills install agents
```

This installs the Curio skills under `~/.agents/skills/`. Remove them with `curio skills uninstall agents`, or remove every CLI-managed Curio skill installation with `curio skills uninstall --all`.

Codex CLI

For the Codex CLI (`codex`), install skills in the shared folder:

```
curio skills install agents
```

Or install them only for Codex:

```
curio skills install codex
```

The second command installs `curio` and `curio-query` under `~/.codex/skills/`. Launch Codex and type `$curio` or `$curio-query` to invoke one directly. Typing `$` displays installed skill completions, and `/skills` lists or manages them. Codex can also choose a matching skill automatically.

Remove the Codex-specific installation with `curio skills uninstall codex`.

ChatGPT Desktop

The ChatGPT desktop app offers **Chat**, **Work**, and **Codex** modes.

> **Which mode should you use?**
> - **Chat** mode cannot interact with the Curio app or invoke its installed skills.
> - Choose **Work** to use Curio skills with your projects—for example, to search, inspect, organize, modify, or produce deliverables.
> - Choose **Codex** for software development with codebase context and developer tools—for example, to write Python scripts that automate Curio workflows.

Install the skills in the shared folder:

```
curio skills install agents
```

Or install them only for ChatGPT's Codex mode:

```
curio skills install codex
```

Verify the installation in ChatGPT by opening **Settings**, choosing **Plugins**, and selecting **Skills**. The Curio skills should appear when installed in either folder.

ChatGPT Web

ChatGPT on the web can connect to Curio's local stdio MCP server through OpenAI's Secure MCP Tunnel. This avoids exposing a public inbound port on your Mac, but ChatGPT can still request Curio data through the tools you allow.

Unsupported feature
Curio does not support configuring or troubleshooting OpenAI's Secure MCP Tunnel. It may involve your organization's infrastructure, security policies, or firewall. Consult OpenAI's documentation for assistance.

How the Tunnel Works

1. Curio provides `/Library/Application Support/CurioCLI/bin/curio mcp-server`.
2. OpenAI's `tunnel-client` runs on your Mac and launches that command over stdio.
3. The client opens an outbound HTTPS connection to OpenAI's tunnel service.
4. ChatGPT sends MCP requests through the tunnel, which forwards them to Curio locally.

Requirements

- The current Curio CLI.
- OpenAI access to Secure MCP Tunnel and ChatGPT custom connectors.
- A tunnel created in OpenAI Platform.
- A runtime OpenAI API key permitted to use that tunnel.
- OpenAI's `tunnel-client` on the Mac that can run Curio.
- The Curio app running for current-selection, current-idea-space, navigation, import/export, or property-change workflows.

Read-only offline queries against a `.curio` file can use MCP tools that expose `project_path`. For selection-based work or project changes, run `tunnel-client` as your normal logged-in Mac user while Curio is open.

Setup Outline

Follow OpenAI's Secure MCP Tunnel documentation to create the tunnel and install `tunnel-client`. Point the local stdio profile at Curio's installed MCP server:

```
export CONTROL_PLANE_API_KEY="sk-..."

tunnel-client init \
  --sample sample_mcp_stdio_local \
  --profile curio \
  --tunnel-id tunnel_0123456789abcdef0123456789abcdef \
  --mcp-command "/Library/Application Support/CurioCLI/bin/curio mcp-server"
```

Validate and run it:

```
tunnel-client doctor --profile curio --explain
tunnel-client run --profile curio
```

Keep `tunnel-client run --profile curio` running while ChatGPT uses the connector. In ChatGPT's connector settings, create a custom connector, choose **Tunnel**, and select or paste its tunnel ID.

Start verification with read-only Curio tools such as status, fields, docs, and query. Enable tools that can change state—such as set, import, export, navigate, create, move, or batch—only when you are comfortable allowing those actions against local Curio projects.

Claude Code CLI

For the Claude Code CLI (`claude`), install the skills with:

```
curio skills install claude
```

This installs `curio` and `curio-query` under `~/.claude/skills/`. Launch Claude Code and type `/curio` or `/curio-query` to invoke a skill, or `/skills` to list installed skills. Claude Code can also choose a matching skill automatically.

Remove them with `curio skills uninstall claude`, or remove every CLI-managed installation with `curio skills uninstall --all`.

Claude Desktop

Claude Desktop has three tabs with different integration behavior:

- The **Code** tab behaves like Claude Code CLI and reads `~/.claude/skills/`.
- The **Chat** and **Cowork** tabs connect over MCP through the Curio Claude Connector.

Code Tab

Install the CLI-oriented skills just as you would for Claude Code:

```
curio skills install claude
```

The Code tab can then call `curio` directly without an MCP layer.

Chat and Cowork Tabs

Install the packaged Curio Claude Connector:

```
curio claude install
```

This creates an `.mcpb` connector package, opens it in Claude Desktop, and lets Claude install Curio's tool and prompt metadata. No `claude_desktop_config.json` edit or manual skill upload is required.

To create the package without opening it immediately:

```
curio claude package
curio claude package —-output ~/Desktop/curio.mcpb
```

The package command reveals the `.mcpb` file in Finder by default. Add `--no-reveal` when scripting package creation.

Troubleshooting the Claude Connector

Verify the CLI and connector prerequisites with:

```
curio claude doctor
```

The command checks the installed `curio` path, connector resources, Claude Desktop availability, old `mcpServers.curio` entries, and whether the MCP server answers an `initialize` request.

Claude Desktop groups Curio tools into read-only and write/delete categories. You can set tools such as **Query**, **Get**, **Docs**, and **Status** to **Always Allow** while leaving tools such as **Set**, **Import**, **Export**, and **Navigate** at **Needs Approval**.

To uninstall the packaged connector:

1. In Claude Desktop, click **Customize** in the sidebar.
2. Click **Connectors**.
3. Select the Curio connector.
4. Click **Uninstall**.

> **Uninstall from Customize**
> Do not uninstall the packaged connector through **Settings > Connectors**. That path may not remove it reliably; use **Customize > Connectors**.

Manual Claude Desktop Configuration

The packaged connector is preferred. For the older manual setup, edit `~/Library/Application Support/Claude/claude_desktop_config.json` and add:

```
{
  "mcpServers": {
    "curio": {
      "command": "/Library/Application Support/CurioCLI/bin/curio",
      "args": ["mcp-server"]
    }
  }
}
```

If the file already contains `preferences` or other top-level objects, add `mcpServers` beside them with valid JSON punctuation.

To remove a manual configuration, quit Claude Desktop, remove the `curio` entry from `mcpServers`, and relaunch. Remove the `mcpServers` key too if it is then empty.

> **Edit the configuration directly**
> Do not use the Settings UI's **Disconnect** button for a classic manual `mcpServers.curio` entry. The UI can report "Invalid server ID format" because the entry has a name rather than a UUID or `mcpsrv_*` ID.

Antigravity CLI

For Antigravity CLI (`agy`), formerly Gemini CLI (`gemini`), install the skills with:

```
curio skills install agy
```

This installs them under `~/.gemini/skills/`. Launch `agy` and type `/curio` to see the Curio skills or `/skills` to list installed skills.

Remove them with `curio skills uninstall agy`, or remove every CLI-managed installation with `curio skills uninstall --all`.

Antigravity Desktop

Antigravity Desktop uses a separate skill directory:

```
curio skills install antigravity
```

This installs the skills under `~/.antigravity/skills/`. Remove them with `curio skills uninstall antigravity`.

Working with AI Agents

Once an integration is installed, begin with a first prompt to see how Curio and the agent work together. Many focused tasks can then be requested with a single ordinary-language instruction. For more complex, multi-step work, use the planning workflow that follows.

First Prompts

For agents using the Curio skills, such as ChatGPT Work or Codex, Codex CLI, Claude Code, Claude Desktop's Code tab, and Antigravity:

> What can you do with the Curio skills available to you? Summarize what you can do with my Curio projects.

For an MCP client such as Claude Desktop's Chat or Cowork tab:

> What Curio MCP tools, prompts, and resources are available? Summarize what you can do with my Curio projects.

After the agent confirms its capabilities, try a read-only request:

> Check whether Curio is running, then summarize the selected figure or active idea space without changing anything.

Or explore a project file without opening it in Curio:

> Inspect `~/Documents/MyProject.curio` in read-only mode. Summarize its sections, major idea spaces, and active tasks.

For Complex Workflows: Plan, Refine, Then Act

Some outcomes involve enough decisions or coordinated changes that they cannot be captured reliably in one instruction. You still do not need to know Curio's commands or specify every implementation detail yourself. Describe the outcome, then let the agent use Curio's published skills to develop the command-level plan with you:

1. **Describe the desired outcome in ordinary language.** Explain what you want to accomplish, which project or selection is involved, and any important constraints.
2. **Ask the agent to propose a plan without changing anything yet.** The agent can inspect available Curio capabilities, identify the necessary steps, and surface questions or assumptions before taking action.
3. **Refine the plan conversationally.** Add details, correct assumptions, change the sequence, or ask the agent to make the plan more thorough. Continue until the proposed result and workflow look right.
4. **Say "Do it."** This clearly tells the agent that the planning phase is complete and it may carry out the approved workflow.

This approach lets you begin with a simple goal while the agent expands it into a detailed, reliable sequence of skills-based Curio commands. It also gives you an opportunity to review the scope and catch misunderstandings before anything changes.

For example:

> I want to prepare this Curio project for a screencast that demonstrates how research moves from an inbox into an organized presentation. Propose a detailed plan using the Curio skills, but do not change anything yet.

After reviewing the response, refine the plan in the same conversation. When you are satisfied, reply:

> Do it.

Prompting Tips

- **Name the scope.** Say whether to use the running Curio app, the current selection, the active idea space, or a specific `.curio` project path.
- **State whether changes are allowed.** Ask for read-only inspection when you do not want modifications. For changes, say whether the agent may act immediately or should propose a plan first.
- **Ask for a preview.** For bulk edits, request the matching item count and a short sample before approving the mutation.
- **Describe the result, not the command.** The installed skills teach the agent the command syntax. A request such as "find overdue active tasks, group them by project, and summarize blockers" is usually enough.
- **Be explicit about saving.** Ordinary imports and other live mutations can be grouped without forcing a save after each command; ask the agent to save the affected Curio project once after the completed workflow.
- **Use stable targets.** For repeatable automation, ask the agent to use UUIDs or identifiers after discovery instead of relying only on titles.
- **Separate inspection from action.** A two-step workflow—first inspect, then approve changes—makes large or ambiguous jobs easier to verify.

- **Keep approvals for risky tools.** In MCP clients, consider automatic approval only for read-only tools and require confirmation for tools that create, update, move, import, export, or navigate.

Example Workflows

Find all active tasks due in the next seven days, group them by tag, and summarize the three most urgent items. Do not change the project.

Examine the current selection, explain which properties can be changed, and wait for my approval before editing anything.

Find figures tagged `Research/Inbox`, show me a five-item sample, then ask whether I want you to move the matches into the Research idea space.

Create a section named “Client Work” with an idea space for each client in this list. Show the proposed structure before creating it, then save the project when finished.

List every file-backed asset in this project and produce a manifest with its title, section path, Organizer path, and asset file path. Do not modify any files.

Help me write and test a reusable Curio query for overdue, incomplete, high-priority tasks. Explain the query before running it.

Creating Standalone Scripts

An AI agent can also turn a recurring Curio workflow into a standalone shell or Python script that you save to disk and run whenever you need it. Describe the result you want in ordinary language; the agent can use Curio’s skills to choose the appropriate commands, parse their JSON output, handle errors, and document how to run the finished script.

> **The agent creates the script; the CLI runs it**
> Once the script has been created, it calls the `curio` CLI directly. Running it does not require ChatGPT, Codex, Claude, an MCP connection, or another AI agent. The agent is only involved while creating, testing, or later improving the script.

The finished script becomes an ordinary reusable automation. You can launch it yourself, schedule it with tools such as `launchd`, or call it from another workflow using only its normal runtime—such as the shell or Python—and the installed Curio CLI.

The script can be as simple as monitoring project backup folders and reporting missing or stale backups, or as sophisticated as querying several Curio projects and generating an interactive dashboard. For example:

> Write a Python script that checks my Curio project backup folder, reports projects without a recent backup, and exits with an error if any backup is more than seven days old. Save the script to disk and include usage instructions.
>
> Build a local dashboard that queries my Curio projects for active tasks, upcoming due dates, and project status. Propose the data model and interface first, without creating files yet.

A coding agent such as Codex or Claude Code is especially useful for this work because it can create and refine the script directly in a working folder, run it, inspect failures, and add tests. If the result includes a web interface, a coding agent with browser tools may also be able to open the dashboard, exercise its controls, and verify the rendered output.

Start with read-only testing
When possible, have the agent develop and test against a copied project or Curio's read-only `--project` mode first. Before allowing a script to change a live project, ask the agent to show which commands will mutate data, add clear error handling, and verify that the intended project is targeted. Live scripts that make changes should finish with `curio save`.