

Blender MCP Bridge

Drive Blender with AI — scene building, animation, Grease Pencil, previews and renders, from a desktop app that needs no terminal.

Launcher v2.11.0

MCP Server v2.2.0

Add-on v2.1.0

Protocol v3

~38 tools

Who this manual is for. Artists, animators and small studios who want an AI assistant (Claude, ChatGPT, Cursor, Antigravity) to operate Blender directly — and anyone installing or supporting the bridge for them. The recommended path uses the **Dork Blender MCP** desktop launcher. Manual, developer-style setup is covered for those who prefer it.

Applies to: Blender 3.6+ (Blender 4.3+ recommended — Grease Pencil v3) · Windows, macOS, Linux
Companion files: `blender_addon.py` · `mcp_server.py` · `launcher/` · `docs/REMOTE_HTTP.md` · `docs/ASSET_TRACE.md`

Contents

1	What Blender MCP Bridge is	3
2	System requirements and file map	3
3	Quick start — the launcher (recommended)	4
4	The launcher window, control by control	5
5	Manual installation (without the launcher)	6
6	Connecting AI clients	7
7	Remote access for web LLMs (ChatGPT / Claude web)	8
8	Everyday workflow	9
9	MCP tools reference — all ~38 tools	10
10	Grease Pencil and 2D animation	11
11	Tracing an image into a Grease Pencil asset	12
12	Jobs, checkpoints, previews and renders	12
13	Configuration reference	13
14	Security model — read this once	14
15	Troubleshooting	14
16	Limitations, updating and uninstalling	15
A	Appendix — protocol v3 and version matrix	15

How to read this manual. Start with Chapter 3 and you will be connected in minutes. Chapters 9 and 15 are the two you will return to most: keep them bookmarked. Code blocks can be copied exactly; replace only the paths and tokens marked in angle brackets.

Conventions

- **Start / READY / Stop** in bold refer to launcher controls and states, spelled exactly as on screen.
- `monospace` is a file name, command, environment variable or tool name.
- *Blender > Preferences* means a menu path inside Blender. *N-panel* is the Sidebar toggled with **N** in the 3D Viewport.

What Blender MCP Bridge is

Blender MCP Bridge connects an AI client to a live Blender session. You describe the result in plain language; the AI calls structured tools that inspect the scene, create and modify objects, set up materials, cameras and lights, animate, and render — while you watch it happen in Blender.

Three pieces work together:

```
AI client (Claude / ChatGPT / Cursor / Antigravity)
| MCP over stdio (or Streamable HTTP in remote mode)
mcp_server.py - FastMCP, ~38 tools, thin wrappers
| localhost TCP 127.0.0.1:9876, protocol v3
| hello handshake + token on every request
blender_addon.py - inside Blender (stdlib + bpy only)
| listener thread -> exec queue (32) -> main-thread pump
| job queue (16) -> one job at a time
Supervised by: python -m launcher / Dork Blender MCP.exe
| detects Blender, starts everything, reconnects
```

- **The add-on** (`blender_addon.py`) runs inside Blender. A listener thread accepts connections but never touches `bpy` ; work is queued and executed on Blender's main thread, where Blender requires it.
- **The MCP server** (`mcp_server.py`) exposes the tools to the AI client. By default it speaks MCP over stdio and has no network surface of its own.
- **The launcher** (the desktop app) detects Blender, installs the add-on copy, manages the auth token, starts Blender and the server, verifies the connection with a protocol handshake, and keeps it alive. Customers never open a terminal.

Every connection opens with a **hello capability handshake**: if the client and bridge disagree on protocol version, you get an `incompatible_version` error naming both sides immediately — never a confusing failure mid-session. Every mutating tool returns a **scene diff** (objects added and removed), and objects are addressed by a **stable uid** that survives renames, so a stale reference fails cleanly instead of acting on the wrong object.

System requirements and file map

Requirement	Details
Blender	3.6 or newer. 4.3+ strongly recommended: Grease Pencil stroke helpers and the asset tracer need Grease Pencil v3. Verified against Blender 4.3 and 4.5 LTS.
Operating system	Windows 10/11, macOS, Linux. The frozen <code>Dork Blender MCP.exe</code> is Windows; macOS/Linux run from source (below).
Python (source installs)	3.10+. The add-on itself needs nothing beyond Blender's bundled Python. The launcher is stdlib-only (tkinter).
MCP client	Claude Desktop, Cursor, Claude Code, Antigravity, or a web LLM via remote mode (Chapter 7).
Disk / memory	Budgets guard heavy work (Chapter 13). Keep ~1 GB free beyond any render or checkpoint output.

Files you will meet

File	What it is
<code>blender_addon.py</code>	The Blender add-on. Single file, stdlib + bpy only.
<code>mcp_server.py</code>	The FastMCP server: stdio by default, <code>--transport http</code> for remote mode.
<code>mcp_http.py</code>	Bearer-gated Streamable HTTP transport for web LLMs.
<code>common.py</code>	Shared helpers: protocol, token files, danger scan, budgets, retention.
<code>asset_trace_core.py</code>	Pure-numpy image tracer embedded into bridge scripts (Chapter 11).
<code>launcher/</code>	The desktop launcher package.
<code>docs/REMOTE_HTTP.md</code> , <code>docs/ASSET_TRACE.md</code>	Deep guides summarised in Chapters 7 and 11.

Quick start — the launcher (recommended)

This is the path the product is built around. Four steps, no terminal, no manual token handling.

- 1 **Get the app.** Download `Dork Blender MCP.exe` (no Python needed) — or, from source, run `python -m launcher` from the project folder. Developers can build the exe with `build_exe.bat` on Windows.
- 2 **Press Start.** Opening the app only shows the window (state: STOPPED) — nothing connects until you press **Start**. The launcher then detects Blender, prepares the token, probes port 9876, launches or attaches to Blender, waits for the bridge (up to 45 s), handshakes, and starts the MCP server.
- 3 **Wait for READY.** Three green dots — **Blender, Bridge, MCP Server** — mean the full chain is verified. If a step fails you get a plain-language error and a **Retry** button instead of a silent hang.
- 4 **Connect your AI client.** Press **Copy client config** and paste it into your client (Chapter 6 shows where each client keeps its config). Then just ask: *"Create a realistic modern kitchen."*

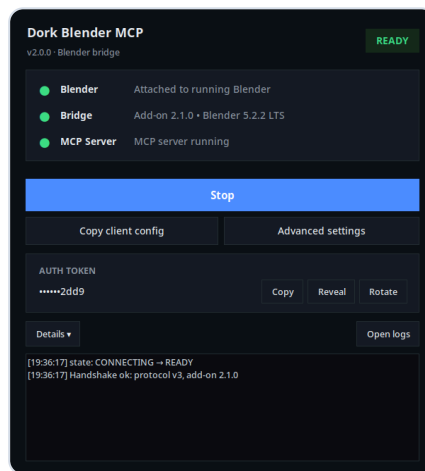


Figure 1 — The launcher at READY: Blender, Bridge and MCP Server all verified.

First-run note. On first Start, the launcher copies `blender_addon.py` into Blender's add-ons folder. If Blender asks, enable **"MCP Blender Bridge v2"** under *Edit > Preferences > Add-ons* once. After installing a new Blender version later, press Start again so the add-on is copied into the new version's folder.

One-click autostart. Blender instances the launcher starts itself receive `DORK_MCP_AUTOSTART=1`, so the bridge server inside Blender starts automatically — that is why you normally never touch the N-panel. If you opened Blender yourself, start the bridge manually (Chapter 5, step 2) and press Retry; the launcher attaches to the running bridge instead of launching a second Blender.

The launcher window, control by control

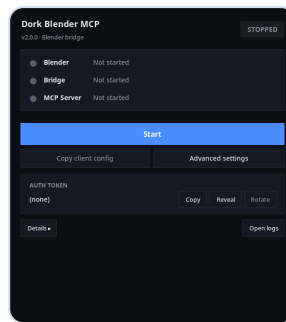


Figure 2 — Just opened: STOPPED. Nothing connects until Start.

Status and the main button

- **Status dots (Blender / Bridge / MCP Server)** show each link in the chain separately, so you can see exactly where a problem sits.
- **Start** becomes **Stop** while running and **Retry** after an error. It is disabled during startup so a double-click cannot spawn duplicates.
- **Stop** stops the MCP server and monitoring and leaves Blender open. Only Blender processes the launcher itself started are ever terminated — a Blender you opened yourself is never killed.
- **Copy client config** copies a ready-to-paste MCP server entry for Claude Desktop, Cursor or Antigravity (Chapter 6).

The token card

The bridge token is created on first Start and shared by add-on, server and launcher (file mode `0600`), displayed masked with **Copy / Reveal / Rotate**. **Rotate** is atomic — old token dies server-side at once; external configs then need a fresh **Copy client config**.

The Remote Access card

Start remote launches the bearer-gated HTTP endpoint for web LLMs; **Start tunnel** runs Cloudflare Tunnel and shows the public `https://...trycloudflare.com` URL; **Copy web config** copies the URL plus `Authorization: Bearer` header. Chapter 7 walks through it. **Stop remote** (or the main Stop) shuts down both the HTTP server and the tunnel.

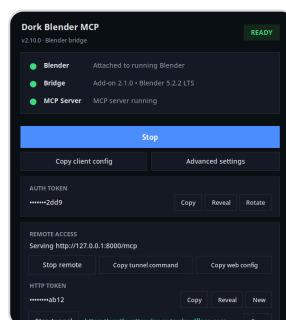


Figure 3 — Remote Access card (v2.10+): remote server, tunnel, and HTTP token controls.

Advanced settings, logs, single instance

- **Advanced settings:** bridge port, startup timeouts, remote HTTP port/path, Blender path, kill switch and log level (Figure 4). Changes apply on the next Start.
- **Logs** are collapsed by default (Details toggle). Duplicates are rate-limited; errors show friendly text.
- **Single instance:** a lock file prevents two launchers fighting over the same bridge. Config lives in the per-OS config dir (Chapter 13) — never beside the exe.

Manual installation (without the launcher)

Use this path if you run from source, are on macOS/Linux, or are wiring the bridge into your own tooling. Four steps.

Step 1 — Install the Blender add-on

1. In Blender: *Edit > Preferences > Add-ons > Install from Disk* (older builds: *Install...*).
2. Select `blender_addon.py` and enable **MCP Blender Bridge v2** (System category).

Manual copy alternative — place `blender_addon.py` in the add-ons folder, then enable it: Windows `%APPDATA%\Blender Foundation\Blender\<ver>\scripts\addons\`; macOS `~/Library/Application Support/Blender/<ver>/scripts/addons/`; Linux `~/.config/blender/<ver>/scripts/addons/`.

Step 2 — Start the bridge inside Blender

1. Open Blender and keep it open — the bridge drives the live session.
2. 3D Viewport > press **N** > **MCP Bridge** tab > set the port (default `9876`) > **Start Server**.
3. The panel shows **RUNNING** with the bound address. Use **Copy Auth Token** here — the server needs it next.

Step 3 — Install the MCP server

```
cd blender-mcp-v2
python -m venv .venv
# Windows: .venv\Scripts\activate | macOS/Linux: source .venv/bin/activate
pip install -r requirements.txt
export BLENDER_MCP_TOKEN=<paste token> # Windows: set BLENDER_MCP_TOKEN=<paste>
```

On the same machine you may skip the env var: the server finds the token file Blender wrote (Chapter 13). You also need Blender's executable for headless renders — the launcher detects it automatically; manual setups rely on `blender` being on PATH: Windows `C:\Program Files\Blender Foundation\Blender 4.5\blender.exe`; macOS `/Applications/Blender.app/Contents/MacOS/Blender`; Linux `/usr/bin/blender`.

Step 4 — Verify before connecting a client

Start Blender + bridge first, then (once a client is configured, Chapter 6) run the smoke test: `ping`, then `get_scene`, then `preview_scene`. If `ping` returns `bridge_unavailable`, the bridge is not running — re-check steps 1–2.

Port already in use? If something else answers on 9876, pick another port in the N-panel *and* in the client env (`BLENDER_MCP_PORT`), or change it once in the launcher's Advanced settings. The bridge refuses non-loopback peers and never binds `0.0.0.0` — that is by design (Chapter 14).

Connecting AI clients

Every client points at `mcp_server.py`. With the launcher, press **Copy client config** and paste — the snippets below are what it copies, shown so manual setups can adapt them.

Claude Desktop

Config file: Windows `%APPDATA%\Claude\claude_desktop_config.json`; macOS `~/Library/Application Support/Claude/claude_desktop_config.json`.

```
{
  "mcpServers": {
    "blender": {
      "command": "/path/to/blender-mcp-v2/.venv/bin/python",
      "args": ["/path/to/blender-mcp-v2/mcp_server.py"],
      "env": { "BLENDER_MCP_TOKEN": "<paste>" }
    }
  }
}
```

With the frozen exe, `command` is `Dork Blender MCP.exe` and the launcher adds the hidden `--mcp-server-child` flag itself — Copy client config produces exactly that entry.

Cursor

Same shape in `~/.cursor/mcp.json` under `"mcpServers"`.

Claude Code (CLI)

```
claude mcp add blender --env BLENDER_MCP_TOKEN=<paste> -- \
/path/to/blender-mcp-v2/.venv/bin/python /path/to/blender-mcp-v2/mcp_server.py
```

Antigravity

Paste the copied entry under `mcpServers` in `mcp_config.json`. For Antigravity's remote (web) mode, use the `serverUrl` + headers form in Chapter 7 instead.

Restart the client after editing its config, then start Blender with the bridge running. Good first prompts once connected:

- *"What is in the scene right now?"* (exercises `get_scene`)
- *"Show me a preview."* (`preview_scene` returns an image)
- *"Create a realistic modern kitchen."* (full build: objects, materials, lights, camera)

Token hygiene. The token in these configs is a real credential for your Blender session. Do not commit it to git, and rotate it (launcher Rotate, N-panel, or the `rotate_token` tool) if it leaks — rotation invalidates the old token immediately.

Remote access for web LLMs (ChatGPT / Claude web)

A browser tab cannot spawn the local exe, so web LLMs connect over HTTPS instead. The launcher serves the **same tools** over MCP's Streamable HTTP transport, bearer-gated, and Cloudflare Tunnel gives it a public HTTPS URL. Note the mental model: it is the *LLM provider's servers* that connect to your URL — your machine must be reachable over HTTPS only for as long as the tunnel runs.

The GUI way (recommended)

1. Press **Start** in the launcher and wait for **READY**.
2. In the **Remote Access** card, press **Start remote**. On first use the launcher generates a separate HTTP Bearer token, stored in a `0600 http_token` file — independent of the bridge token.
3. Press **Start tunnel** (one-time prerequisite: `cloudflared` on PATH). The public `https://<random>.trycloudflare.com` URL appears in the card.
4. Press **Copy web config** and paste it into the web LLM's MCP connector: server URL `https://<random>.trycloudflare.com/mcp` with header `Authorization: Bearer <http-token>`.

ChatGPT: Settings > Connectors > add custom MCP server. **Claude web:** Connectors > add custom connector.

Antigravity remote: `{"serverUrl": "https://<tunnel>/mcp", "headers": {"Authorization": "Bearer <http-token>"}}`.

The manual way

```
python -c "import secrets; print(secrets.token_urlsafe(32))" # generate HTTP token
export BLENDER_MCP_TOKEN=<bridge token>
export DORK_MCP_HTTP_TOKEN=<the token you just generated>
python mcp_server.py --transport http --http-port 8000
# frozen exe equivalent: set the same env vars, then
# "Dork Blender MCP.exe" --mcp-http-child
cloudflared tunnel --url http://localhost:8000
```

Options: `--http-host` (default `127.0.0.1` — keep it unless the proxy runs elsewhere), `--http-port` (default `8000`), `--http-path` (default `/mcp`); also settable via `DORK_MCP_HTTP_HOST/PORT/PATH/TOKEN`. Port and path are editable in the launcher's Advanced settings and apply at the next remote start.

Security behaviour (enforced, tested). No HTTP token, no server — it refuses to start. Every request needs the Bearer header (constant-time compare); missing or wrong returns `401` with nothing leaked. The token never appears in logs. **Never expose the raw http:// port to the internet** — always front it with the tunnel or another HTTPS proxy. Blender itself stays localhost-only; only the MCP endpoint is remote. The HTTP token rotates independently: **New** next to HTTP TOKEN regenerates it and restarts the HTTP server; the bridge **Rotate** button restarts the HTTP child with the fresh bridge token automatically.

Quick local check: `curl -i http://127.0.0.1:8000/mcp` with the Bearer header and an `initialize` JSON-RPC body returns a session id; without the header, `401`. Full details: `docs/REMOTE_HTTP.md`. Honest scope: no built-in TLS (the tunnel provides it), no OAuth — a single shared Bearer token, right for personal and small-team use.

Everyday workflow

The tools are designed around a rhythm that keeps AI work predictable and reversible:

1. **Inspect first.** `get_scene` and `inspect_object` before generating code, so scripts target objects and layers that actually exist. Objects come back with stable uids — use them.
2. **Mutate, then read the diff.** Every create/modify/delete answers with what was added and removed. If an object was deleted or replaced since inspection, you get `stale_reference` — re-inspect rather than guessing.
3. **Checkpoint before anything big.** `create_checkpoint` is the reliable rewind (Chapter 12). `undo` is a single Blender undo step and may not exist headless.
4. **Preview cheap, render once.** `preview_scene` renders a fast low-sample Eevee image inline. `render_animation` runs as a background headless job so Blender stays responsive.
5. **Prefer structured tools; keep the escape hatch.** The ~37 structured tools cover most work. `execute_blender_python` runs arbitrary `bpy` code for everything else — it is scanned for dangerous calls first (Chapter 14).

A first animation, end to end

```
# What the AI does under the hood (you just prompt for it):
get_scene()                                # what exists?
execute_blender_python(code="..." )       # draw strokes / keyframes
preview_scene(width=640, samples=8)         # fast visual check
render_animation("/shots/shot01/frame_####.png", 1, 48)
```

Typical session prompts that map cleanly onto the tools: *"Add a bouncing ball across 24 frames"*, *"Light it like a studio product shot"*, *"Checkpoint this, then try a low-angle camera"*, *"Trace this boat image into a reusable asset"* (Chapter 11).

Long work belongs in jobs. Jobs occupy Blender's main thread; cancellation is cooperative via `job_should_cancel()`, and code stuck in one long C call cannot be force-killed. Animation renders avoid the problem entirely by running in a separate `blender -b` process that *can* be terminated.

MCP tools reference — all ~38 tools

Signatures are abbreviated; every tool returns JSON. Mutating tools include a scene diff. `ref` accepts a stable uid (preferred), a name, or a scene path.

Group / Tool	Purpose and key inputs
Connection <code>ping</code>	Liveness + handshake check. First call in any smoke test.
<code>rotate_token</code>	Issue a fresh bridge token; the old one dies immediately.
Code / jobs <code>execute_blender_python</code>	Run arbitrary Python/bpy in Blender. AST danger-scanned; pass <code>confirm_dangerous=true</code> only when you mean it (logged, Chapter 14).
<code>start_job</code> · <code>job_status</code> · <code>job_result</code> · <code>cancel_job</code>	Long-running code as a managed job: start, poll, fetch result, cooperatively cancel.
Scene <code>get_scene</code>	Scene snapshot: objects, frames, fps, resolution, engine, camera, Grease Pencil layers. <code>detail</code> : summary/full.
<code>inspect_object</code>	Deep look at one object by uid/name.
<code>get_blender_capabilities</code>	Blender/add-on/protocol versions and feature flags (e.g. Grease Pencil v3).
Objects <code>create_object</code> · <code>modify_object</code> · <code>delete_object</code>	Create (cube, plane, ...), transform/rename, delete — all diff-reporting, uid-safe.
Materials <code>create_material</code> · <code>assign_material</code> · <code>inspect_material</code>	Principled materials with base colour; assignment by ref; inspection.
Nodes <code>inspect_node_tree</code> · <code>node_tree_add_node</code> · <code>node_tree_link</code> · <code>node_tree_set_value</code> · <code>create_geometry_nodes</code>	Shader, compositor and Geometry Nodes editing: inspect, add, link, set values, scaffold a GN tree.
Camera / light <code>set_camera</code> · <code>set_lighting</code>	Camera placement/lens; lighting presets (e.g. studio) with energy.
Animation <code>add_keyframes</code> · <code>inspect_animation</code> · <code>add_constraint</code> · <code>rig_character</code>	Keyframes per property/frame; animation inspection; constraints; template character rigger (fixed proportions, auto-weights — a starting point, not a finished rig).

MCP tools reference (continued)

Group / Tool	Purpose and key inputs
Assets import_asset · export_asset	Import/export (fbx, ...), confined to BLENDER_MCP_ASSET_DIRS .
trace_asset_to_grease_pencil	Image → layered Grease Pencil asset. Chapter 11.
Files save_blend · open_blend · create_checkpoint · list_checkpoints · restore_checkpoint · undo	File lifecycle plus named checkpoints (the reliable rewind).
Render preview_scene	Fast Eevee preview returned as an image (default 640 px, 8 samples).
render_animation	Background headless render of a frame range to frame_####.png -style output; returns a job/pid immediately.

Tool count is 38 including the tracer; remote HTTP mode serves the same set. Exact parameter schemas are exposed by the server itself — ask your client to list tools, or read the docstrings in mcp_server.py .

Grease Pencil and 2D animation

The bridge is built for Grease Pencil (GP) work — the workflow behind productions like the 45-second 2D short *The Paper Boat*: the boat sits *in* the water (waterline crossing the hull, ripples, contact shadow), never pasted on top, and flat single-plane "sticker" placement is treated as a defect.

- **GP v3 helpers** are pre-exposed in every `execute_blender_python` namespace: `gp_get_or_create`, `gp_layer`, `gp_keyframe`, `gp_draw_stroke` (points, radius, material), plus `set_frame_range`, `render_setup` and `scene_state`. They need Blender 4.3+; older builds should use raw `bpy`.
- **Draw, keyframe, preview.** Strokes are drawn per layer/frame, keyframed, then checked with `preview_scene` before any full render. Eevee previews are fast and low-sample by design — final quality comes from `render_animation`.
- **Depth reads on screen** through line weight, cel shading, contact shadows, reflections and layered backgrounds — prompt for these explicitly; the tools can build all of them.

Tracing an image into a Grease Pencil asset

`trace_asset_to_grease_pencil` turns a flat 2D reference image (PNG/JPG) into a reusable, layered Grease Pencil asset inside Blender — no manual tracing, no Pillow/OpenCV installs. The traced asset is how a design stays identical across every shot with zero drift.

1. Save the reference image under `BLENDER_MCP_ASSET_DIRS` (default `~/blender_mcp_assets`).
2. Call `trace_asset_to_grease_pencil(image_path, asset_name, ...)` with a name like `BoatHero`.
3. The asset lands as `ASSET_<asset_name>` in the **DorkAssets** collection, marked as a Blender asset, one GP layer per colour region (`ASSET_<name>_00` , `_01` , ...). Instance it in later shots instead of re-drawing.

How it works. The pure-numpy core (`asset_trace_core.py`) quantises colours (median-cut, largest-gap splitting), extracts boundary loops (vectorised marching squares), simplifies them (Douglas–Peucker), and skips the border-touching background colour. The server embeds that source into a fixed bridge script — only JSON parameters vary — so it passes the danger scanner without confirmation, and Blender's bundled numpy plus `bpy.data.images.load` do the decoding.

Anti-sticker design. Each colour region gets its own layer with a small baked Z offset (`depth_step` , default 0.03) for real parallax, plus flat fills with an ink outline stroke 0.006 above the fill. The result is a posable object, never axis-locked.

Parameter	Default	Notes
<code>image_path</code>	—	Confined to asset dirs; .png/.jpg/.jpeg, ≤ 25 MiB.
<code>asset_name</code>	—	<code>[A-Za-z][A-Za-z0-9_]{1,64}</code> ; re-running replaces the asset.
<code>target_height</code>	2.0	Asset height in Blender units (0.1–50).
<code>max_colors</code>	8	Colour regions (2–16).
<code>simplify_tolerance</code>	1.5 px	Outline smoothing (0–10; higher = fewer points).
<code>line_width</code>	3.0	Ink stroke radius (0.5–50).

Limits. Best on flat vector-style art; photographs posterise noisily (raise `max_colors` , expect cleanup). Needs Blender in Object mode. Verified end-to-end on Blender 4.5.3 LTS (boat asset → 7 regions, 9 fills + 9 ink strokes, correct orientation, depth offsets, asset marking). Full notes: `docs/ASSET_TRACE.md` — including the GPv3 API facts (data at `bpy.data.grease_pencils_v3` , `drawing.add_strokes(...)` , template material via `grease_pencil_add` , bottom-row-first pixels) that only real-Blender testing revealed.

Jobs, checkpoints, previews and renders

Jobs

`start_job` runs long code under the job system (queue 16, one at a time; exec queue 32). Poll with `job_status`, fetch with `job_result`, cancel cooperatively with `cancel_job`. Overload returns `busy` — nothing is silently dropped. Finished jobs are pruned beyond 100 entries or 24 h.

Checkpoints

`create_checkpoint` / `list_checkpoints` / `restore_checkpoint` are the dependable undo. Retention evicts oldest-first beyond 20 checkpoints or 72 h. `undo` is one Blender step and may not exist headless — checkpoint before risky work.

Previews

`preview_scene` renders Eevee at low samples and returns the image inline — the fast loop for framing, lighting and GP checks.

Renders

`render_animation` snapshots the .blend if dirty (your live session is untouched) and spawns `blender -b` detached, returning immediately with pid and output path. Blender stays responsive; the subprocess can be terminated.

Checkpoints, renders, exports and previews all check free disk first (estimate + 1 GiB headroom) and fail with `insufficient_disk_space` naming free vs required, instead of dying halfway.

Configuration reference

Setting	Default / location
Bridge address	<code>127.0.0.1:9876</code> — localhost only, always. <code>BLENDER_MCP_PORT</code> changes the port; <code>BLENDER_MCP_HOST</code> is ignored unless <code>BLENDER_MCP_REMOTE_MODE=1</code> (loud warning; still token-authed).
Remote HTTP	Port <code>8000</code> , path <code>/mcp</code> ; <code>--http-host/port/path</code> or <code>DORK_MCP_HTTP_*</code> . HTTP token in the <code>0600 http_token</code> file beside the config.
Bridge token	32 random bytes, mode <code>0600</code> , in Blender's config dir (<code>mcp_bridge_token</code>). Resolution order: <code>BLENDER_MCP_TOKEN</code> env, then the token file. Rotate via tool, N-panel or launcher.
Asset directories	<code>BLENDER_MCP_ASSET_DIRS</code> ; default <code>~/blender_mcp_assets</code> + temp dir. Import/export/open/trace are confined here.
Budgets	2M polygons, 10k frames, 33 MP, 8192 px textures. Override via <code>BLENDER_MCP_BUDGETS_JSON</code> or per-call <code>override_budgets</code> . Breaches return <code>budget_exceeded</code> .
Retention	<code>BLENDER_MCP_MAX_CHECKPOINTS</code> (20), <code>BLENDER_MCP_CHECKPOINT_MAX_AGE_HOURS</code> (72), <code>BLENDER_MCP_JOB_HISTORY_LIMIT</code> (100), <code>BLENDER_MCP_JOB_HISTORY_TTL_HOURS</code> (24).
Launcher config	<code>%APPDATA%\Dork Blender MCP\ / ~/Library/Application Support/Dork Blender MCP / ~/.config/dork-blender-mcp.MCP</code> child log: <code>mcp_server.log</code> there.
Wire protocol (v1 folder)	The legacy <code>~/workspace/blender-mcp/</code> pair (<code>blender_server.py</code> + <code>mcp_bridge_server.py</code> , 3 tools, no auth) speaks length-prefixed JSON on the same port. Do not mix it with v2: run one bridge per port.

Security model — read this once

- **Localhost only.** The add-on binds `127.0.0.1`, refuses `0.0.0.0`, and drops non-loopback peers. The MCP server is stdio-only by default; its bridge target is forced to `127.0.0.1`. Remote access exists only through the bearer-gated HTTP mode (Chapter 7) behind HTTPS.
- **Token auth on every request**, with rotation that kills the old token immediately. Anyone with the token can execute Python in Blender — treat it like a password, because it is one.
- **Danger scanning.** `execute_blender_python` and `start_job` are AST-scanned on *both* sides for `subprocess`, `os.system`, `rmtree`, `network` and `ctypes` patterns, and blocked unless `confirm_dangerous=true`. Plain `import os` and collection `.unlink()` are fine; only genuinely dangerous receivers are flagged.
- **Audit log.** Every explicitly confirmed dangerous execution is appended to `mcp_blender_audit.jsonl` (temp dir): timestamp, request/job id, code SHA-256, truncated code, findings, duration, status, affected objects. Clean executions are deliberately not logged.
- **Path confinement and budgets** (Chapter 13) plus disk-space guards bound what file and render operations can do.

The honest bottom line. Full-Python execution in Blender is a deliberate feature — it is the escape hatch that makes the bridge general. The full-Python Blender tool must never be reduced. That power is exactly why the token, the localhost binding and the danger gate exist. Never port-forward the bridge port.

Troubleshooting

Symptom / error	Fix
bridge_unavailable / "Cannot reach Blender bridge"	Blender open? Add-on enabled? Bridge started (launcher Start, or N-panel > MCP Bridge > Start Server)? Port matches everywhere?
auth_error	Token mismatch. Copy Auth Token again (launcher or N-panel), or Rotate and re-copy client config.
"Port 9876 is used by another application" / cannot bind 127.0.0.1:9876	Free the port, or change it in Advanced settings (launcher) / N-panel + client env (manual).
incompatible_version / handshake_required	Protocol skew. Update the add-on and server from the same release; the error names both versions.
busy	Bridge saturated (exec 32 / jobs 16 / 8 connections). Wait and retry.
dangerous_code_blocked	The scan found a risky call. Review it; re-run with confirm_dangerous=true only if you mean it (it will be audit-logged).
stale_reference	Object deleted or replaced since inspection. Re-run get_scene / inspect_object and use the fresh uid.
budget_exceeded	Reduce scope, or pass override_budgets=true deliberately.
insufficient_disk_space	Free space or retarget the output/checkpoint directory.
GP stroke helpers fail	Needs Blender 4.3+ (Grease Pencil v3). Check bpy.app.version ; use raw bpy on older builds.
"Blender executable not found" (renders)	Install Blender on PATH, or point the launcher at the binary (Advanced settings). Manual: full path per Chapter 5.
Launcher: bridge unreachable after Start, Blender opened manually	Start the N-panel server manually, then Retry — the launcher attaches to the running bridge.
Web LLM gets 401	Wrong or rotated HTTP token. Copy the current HTTP TOKEN from the Remote Access card into the connector.

Limitations, updating and uninstalling

Known limitations (honest list)

- Jobs run on Blender's main thread; cancellation is cooperative and a single long C call cannot be force-killed. Renders sidestep this via the headless subprocess.
- `rig_character` is a template rigger (fixed proportions, auto-weights), not a hand-built rig replacement.
- `undo` is one Blender undo step and may not exist headless — checkpoints are the reliable rewind.
- Previews are Eevee at low samples: fast, not final quality.
- Remote mode has no built-in TLS and no OAuth: one shared Bearer token behind the tunnel's HTTPS, suited to personal/small-team use.
- The tracer excels at flat vector-style art; photographs need cleanup (Chapter 11).

Updating

1. Install the new release's exe (or pull the new source).
2. Press **Start** — the launcher re-copies `blender_addon.py` into each Blender version's add-ons folder; re-enable **MCP Blender Bridge v2** if Blender prompts.
3. If versions skew, `incompatible_version` names which side to update — update add-on and server from the same release.

Uninstalling

1. Launcher: **Stop**, quit, delete the exe; remove the config dir (Chapter 13) to drop tokens and logs.
2. Blender: *Preferences* > *Add-ons* > remove **MCP Blender Bridge v2** (or delete `blender_addon.py` from the add-ons folder). The `mcp_bridge_token` file in Blender's config dir can be deleted with it.
3. Clients: remove the `blender` entry from the MCP client config and restart the client.

Protocol v3 and version matrix

Transport. Length-prefixed JSON over TCP (4-byte big-endian length + UTF-8 JSON) on `127.0.0.1:9876`.

Handshake. The client's first message is `hello` with its protocol and client version; the bridge answers with protocol, add-on and Blender versions — or `incompatible_version` naming both sides. Every later request carries the token. Responses include status, result, a scene diff for mutations, and structured error codes (Chapter 15). Inside executed code, `print()` output returns as logs and assigning the global `RESULT` returns structured data.

Component	Version (this manual)
Desktop launcher	2.11.0 (title bar shows it — check it when verifying a running build)
MCP server	2.2.0
Blender add-on	2.1.0 (<code>bl_info</code> (2, 1, 0), minimum Blender (3, 6, 0))
Wire protocol	v3
Runtime deps (source)	<code>mcp~=1.30.0</code> , <code>uvicorn~=0.54.0</code> (HTTP mode only); add-on and launcher are stdlib-only

Test evidence behind this release: 253 pytest cases passing, including live headless-Blender verification of the asset tracer on Blender 4.5.3 LTS. The legacy v1 pair in `~/workspace/blender-mcp/` (add-on 1.0.0, three tools, unauthenticated) is documented in its own README and is superseded by v2 — do not run both on the same port.

Keep this manual with the build it describes. The launcher title bar, the add-on's `bl_info` , and `SERVER_VERSION` in `mcp_server.py` are the three version strings to quote in any bug report, alongside the error code from Chapter 15 and the relevant lines from `mcp_server.log` .