
GaiaOS — Layman's User Guide

1) What GaiaOS actually does (in one paragraph)

GaiaOS runs a small "text server" that **reads a script file**, executes the commands **in order**, and **signals when it's finished** using simple "flag" files. You (or the Raspberry Pi driver) drop a command like `eval something 1.0` into `Control_Panel.ssv`, set the **flag** file to notify GaiaOS, it runs the commands, then flips a **finished** file so you know it's done. Under the hood, it can keep rolling data for inputs ("afferents") and outputs ("efferents"), and it can bucket/normalize inputs into simple bands ("granulation") so even noisy sensor values are easy to reason about. The server reads a file token-by-token and runs actions, then sets `Control_Panel_Finished.ssv` when it completes.

---

2) What you need on disk (make this structure)

Create a working folder with these files and subfolders:

Control_Panel.ssv

Control_Panel_Flag.ssv

Control_Panel_Finished.ssv

Output/returned.ssv

Scripts/ # optional, for reusable scripts

autoexec.ssv # optional, runs on boot

pinout.cfg # only if using Raspberry Pi hardware bridge

- The server **reads** `Control_Panel.ssv` when the flag indicates there's work, and **sets** `Control_Panel_Finished.ssv` to mark it done.
- Most "printouts/returns" land in `./Output/returned.ssv`.
- On boot, it can run `autoexec.ssv` automatically (via the `./Scripts/../autoexec.ssv` default path—quirky but fine).

---

3) Quick install and smoke test (no hardware needed)

Build the server

g++ -std=c++17 -o gaia NT4.cpp

./gaia

The executable constructs the Gaia text server and enters its `run()` loop.

Run a "hello world" command cycle

1. Put this into `Control_Panel.ssv`:

help

exit

2. Tell GaiaOS to run it by creating/updating the flag:

```
echo 1 > Control_Panel_Flag.ssv
```

3. GaiaOS will read the script **sequentially**, execute, then write **1** into `Control_Panel_Finished.ssv`. Check the finished file to confirm the cycle completed.

If nothing happens: make sure the server is running and the file names match exactly. The server polls the **flag** file and, when it sees content (conventionally “1”), it processes `Control_Panel.ssv`.

---

4) The “flag file” idea in plain language

- **`Control_Panel.ssv`** — your to-do list for GaiaOS.
- **`Control_Panel_Flag.ssv`** — your “doorbell.” Put a `1` in there to say “job ready.” (That’s the convention the Pi bridge uses.)
- **`Control_Panel_Finished.ssv`** — GaiaOS rings this bell by writing `1` when it’s done.

The server really does run the control file **token by token** (word by word) in order. That’s why you keep each command and its arguments on one line.

---

5) Common commands you’ll actually use

- `help` — prints help text.
- `eval "<label>" <strictness>` — do a general evaluation pass; start strictness around 1.0–1.5.
- `Train` — use current inputs to update the model.

- `Cogitate <filename> <score>` — evaluate and propose outputs, optionally filtering by score.
- Various “dump”/“view” commands can write details into `Output/returned.ssv`. The server’s return file is defined at `./Output/returned.ssv`.

> Tip: After a cycle, **open `Output/returned.ssv`** to see what came back. That’s your first diagnostic lens.

---

6) Hardware integration with a Raspberry Pi (optional, but handy)

What the Pi driver does

- Reads your **`pinout.cfg`** (one device per line), configures GPIO/I²C/1-Wire, samples each input, and **atomically writes** the numeric value into the file you specify (no half-written files).
- It periodically checks the control-panel flag; if there isn’t an in-flight job, it **drops a command** (default example: `eval Testermon.txt 0.0`) and flips the flag to `1`.

`pinout.cfg` cheat-sheet (line formats)

```
A    <pin> <pin> …   <filepath>   # Read a group of GPIO pins as a single integer → file

E    <pin>           <filepath>   # Drive one output pin based on file contents (0/1)

A1W  <sensor_id>     <filepath>   # 1-Wire temperature sensor (°C) → file

US   <trig> <echo>   <filepath>   # HC-SR04 ultrasonic distance (cm) → file

PH   <i2c_addr_hex> <chan> <filepath>  # pH via ADS1115 (needs slope/offset)
```

The driver validates each line and sets up the right behavior.

Note on pH: you must set calibration voltages for pH 7 and pH 4 so the driver can compute slope/offset. The example shows V7/V4 and the math.

Atomic writes: the driver always writes via temp-file-and-rename so readers never see partial contents. That's by design—borrow the same pattern if you write any files yourself.

Clean shutdown: the driver installs signal handlers and calls `GPIO.cleanup()` on exit. That prevents "stuck" pins after crashes or CTRL-C.

Minimal Pi workflow

1. Fill out `pinout.cfg` with your sensors and the **destination files** (where values should land).

Run:

```
sudo python3 RPi_Driver.py
```

2.
3. Watch it log sensor reads and file writes; it will periodically drop an `eval …` line into `Control_Panel.ssv` and set the flag to `1`.

---

7) How inputs are "made sane" (what granulation means)

For each input stream, GaiaOS can:

- **Remember a short history** at a chosen depth,
- **Bucket the value into ranges** you define (goal band first, then wider ones),
- **Track the direction of change** (delta), and
- Compute a simple **deviation** (which way/how far to correct).

That's wired up through the A/E interface and granulator:

- Set the **depth** (how many time steps you remember).
- **Add granulation bands** starting from the "goal" (tightest) outwards.
- When you **set a value**, it automatically granulates, computes delta, and deviation for you.
- You can query concrete, granulated, delta, and deviation if you need to introspect.

> Practical meaning: "Keep the tank at ~75°F" becomes "we're in the green/yellow/red band and trending up/down," which is way easier to use for rules or learning than raw floats.

---

8) Golden-path checklist for first install

A. Server, no hardware

1. Build and run `gaia`.
2. Put a small script into `Control_Panel.ssv` (e.g., `help`, `exit`).
3. Write `1` to `Control_Panel_Flag.ssv`. GaiaOS reads the file top-to-bottom and sets `Control_Panel_Finished.ssv` when done.

B. Server + Raspberry Pi

1. Wire sensors/actuators and create `pinout.cfg` entries.
2. Run `RPi_Driver.py` (as root if necessary for GPIO/I²C).
3. Confirm it logs file writes, and that you see values showing up in the paths you named. The driver will also write `eval …` to `Control_Panel.ssv` and set the flag to `1` on a loop (by default).

---

9) How to tell “it’s working”

- **After a run request**: `Control_Panel_Finished.ssv` should contain `1`. If it never appears, the server didn’t successfully interpret the script. Check file names and permissions.
- **You see output**: `Output/returned.ssv` contains fresh lines after a command that writes returns.
- **Pi driver logs**: lines about “Configured …” and “wrote … to …” indicate successful reads/writes.

---

10) Troubleshooting — symptoms → fixes

Symptom: You set the flag but nothing runs.

- **Check names and working directory.** Files must be exactly `Control_Panel.ssv` and `Control_Panel_Flag.ssv` alongside the server process.
- **Ensure the flag has content.** Convention is `1`; the server checks for presence and then proceeds.

Symptom: The finished flag never flips.

- Script parse failed. Open `Control_Panel.ssv` and keep each command and its args on one line; the server reads tokens in sequence and will bail on malformed lines.
- On failure, the server emits an error and won’t set a success status; it only writes `Control_Panel_Finished.ssv` on handled paths.

Symptom: Pi driver writes partial data (values look truncated).

- Use the built-in **atomic write** helper. If you wrote your own bridge code, copy this pattern (temp file + `os.replace`).

Symptom: Ultrasonic or pH values read as nonsense.

- Verify `US <trig> <echo> <file>` and `PH <addr> <chan> <file>` line formats in `pinout.cfg`. The parser rejects malformed lines—watch the warnings.
- For pH, re-measure V7 and V4 and update slope/offset math.

Symptom: GPIO pins stay “stuck” after you stop the driver.

- Make sure you didn’t kill it with `-9`. The driver handles SIGTERM/SIGHUP and cleans up GPIO properly.

---

11) Field diagnostics (quick tests you can run)

A. Confirm the server reads and acknowledges jobs

1. Put `help` and `exit` in `Control_Panel.ssv`.
2. `echo 1 > Control_Panel_Flag.ssv`.
3. Check that `Control_Panel_Finished.ssv` becomes `1`.

B. Confirm the Pi driver is writing sensor values

- Start the driver; check its log lines: “Configured …” for each device and “wrote … to …” after each read. Then inspect the destination files you named; they should **change over time**.

C. Sanity-check input normalization

- If you’re using granulation, ensure the **goal band** is registered first, then wider bands. That’s how the granulator interprets sign/magnitude.

---

12) FAQ (shop-floor edition)

Q: Where do I look for results?
`Output/returned.ssv` is the default place many commands write to.

Q: Can I chain scripts?
Yes. Place files in `./Scripts/` and you can call a file like a command token; files can call other files. Use sparingly to avoid spaghetti.

Q: Does it auto-run a startup script?
Yes—`autoexec.ssv` (via a path quirk of `./Scripts/../autoexec.ssv`).

Q: How do I know a job actually ran?
`Control_Panel_Finished.ssv` flips to `1` after a successful interpret of `Control_Panel.ssv`.

---

13) When in doubt, reduce the system

- Test the **server alone** with a tiny script and the flag files.
- Then bring in **one sensor** with a simple `pinout.cfg`, and verify the file gets written by the driver.
- Only after those pass do you add more devices and the more complex scripts.

---

14) One last mental model (so you can reason about errors fast)

- GaiaOS = **file interpreter** + **handshake**. It reads a **to-do file** and toggles a **done file**.
- Pi driver = **hardware** → **files** + **nudger**. It writes sensor numbers atomically and taps GaiaOS on the shoulder with `Control_Panel_Flag.ssv`.

If you keep that picture in mind, installations and diagnoses collapse to a few predictable checks: “Did the files update? Did the flags flip? Did the returns show up?” That’s the heartbeat of the system.

Gaia Shell Manual

GaiaOS is controlled entirely by **scripts** (`.ssv` files) that contain commands and arguments. The server reads these files when you flip the control-panel flag, executes the commands top-to-bottom, and signals completion.

This manual lists the available commands, their usage, and what to expect.

Basics

Script Execution

- GaiaOS reads `Control_Panel.ssv` **token by token**, left to right, top to bottom.
- Each line should contain **one command and its arguments**.
- Execution stops when the file ends, or on a fatal error.

Command Flags

- After GaiaOS finishes a run, it sets `Control_Panel_Finished.ssv` to `1`.
- Many commands write their outputs to `./Output/returned.ssv` (the default return file).

Core Commands

`help`

Usage:

help

Prints the help menu with available commands. Good smoke test.

`exit`

Usage:

exit

Terminates the GaiaOS server loop gracefully.

`Train`

Usage:

Train

Consumes current afferent values (from input files or manually set) and updates the model's memory/weights.

Notes:

- You must have registered afferents before training.
- Training modifies in-memory state; persistence is not automatic yet.

`Cogitate`

Usage:

Cogitate <filename> <score>

Runs an evaluation of inputs and proposes outputs (writes to efferent files).

- `<filename>`: Label or dataset reference.

- `<score>`: Threshold filter; larger values are stricter, smaller more permissive.

Notes:

- Outputs appear in efferent files (`./IO_Files/E/*.e.ssv`) or in `returned.ssv`.
- Designed for “thinking about” inputs and producing corrective actions.

`eval`

Usage:

eval "<label>" <strictness>

Runs a general evaluation pass.

- `<label>`: Descriptive string (e.g. “test1”).
- `<strictness>`: Recommended range ~1.0–1.5. Lower = looser, higher = stricter.

Example:

eval "Testermon.txt" 1.2

Notes:

- A go-to command for cycling the system in practice.
- Outputs to `./Output/returned.ssv`.

IO and State Commands

IO Registration

When you register afferents or efferents (inputs/outputs), GaiaOS creates file pairs:

Input.n.ssv / Input_Flag.n.ssv

Output.n.ssv / Output_Flag.n.ssv

These are used for asynchronous handshakes (like PLC I/O).

Command forms (examples):

register_afferent <index> <name> <depth>

register_efferent <index> <name>

Notes:

- Depth defines how many timesteps are remembered.
- Names are optional but make logs readable.
- After registration, you can set values by writing to the `.ssv` file and flipping the flag.

Depth Control

set_depth <n>

Sets the number of timesteps to remember for afferent histories.
Example: `set_depth 5` means each input remembers 5 values.

Granulation

add_range <min> <max>

Adds a granulation band. Ranges must be declared from tightest goal → broader ones.
The granulator maps values into ±band index:

- Negative = below midpoint,
- Positive = above midpoint.

Always finish with a broad catch-all band.

Introspection & Debugging

TSG

Usage:

TSG

Dumps information about the time-series generator (internal learning structure).
Output goes to `Output/returned.ssv`.

AE

Usage:

AE

Prints the afferent/efferent IO map. Useful for confirming that inputs/outputs are registered correctly.

Scaffold

Usage:

Scaffold

Dumps the scaffold structure (network of learned states).

`NodeNet`

Usage:

NodeNet

Prints the current node network for debugging the learning model.

Script Handling

Including Scripts

You can call a script file by name like it was a command:

myscript.ssv

GaiaOS interprets the file as part of the command sequence.
Scripts can call other scripts, but beware recursion and “dependency hell.”

Autoexec

At startup, GaiaOS looks for:

./Scripts/../autoexec.ssv

and executes its contents before entering the main loop. Use this to bootstrap IO registration and initial training.

Practical Examples

1. Simple evaluation cycle

eval "test_run" 1.0

2. Training after setting inputs

Train

Cogitate inputs.ssv 0.8

3. Setup script (`autoexec.ssv`)

set_depth 5

register_afferent 0 temperature 5

register_efferent 0 heater

Train

Operational Notes

- Commands are case-sensitive (stick to examples).
- Always ensure `Control_Panel_Flag.ssv` is written with `1` before expecting GaiaOS to act.
- Read results from `Output/returned.ssv` after each cycle.
- Strictness tuning: 1.0–1.5 is the “happy zone.” Use <1.0 for exploration, >1.5 for narrow precision.

Gaia Shell Cheat Sheet

File Protocol

- **Control_Panel.ssv** → script of commands
- **Control_Panel_Flag.ssv** → set to 1 to tell GaiaOS to run
- **Control_Panel_Finished.ssv** → GaiaOS writes 1 when done
- **Output/returned.ssv** → default result dump

Core Commands

Command	Usage	Effect	Output
`help`	`help`	Show available commands	Console / returned.ssv
`exit`	`exit`	Stop GaiaOS loop	—
`Train`	`Train`	Train on current inputs	Updates in-memory model
`Cogitate`	`Cogitate <filename> <score>`	Evaluate + propose outputs	Efferent files / returned.ssv
`eval`	`eval "<label>" <strictness>`	General evaluation (strictness ~1.0–1.5)	returned.ssv

---

IO & Setup

Command	Usage	Effect	Files

`register_afferent`	`register_afferent <id> <name> <depth>`	Add input channel	Creates `Input.<id>.ssv` + `Input_Flag.<id>.ssv`
`register_efferent`	`register_efferent <id> <name>`	Add output channel	Creates `Output.<id>.ssv` + `Output_Flag.<id>.ssv`
`set_depth`	`set_depth <n>`	Set memory length	Affects afferent histories
`add_range`	`add_range <min> <max>`	Add granulation band	Appends to granulator ranges

Introspection

Command	Usage	Effect	Output
`TSG`	`TSG`	Show time-series generator info	returned.ssv
`AE`	`AE`	Dump afferent/efferent map	returned.ssv
`Scaffold`	`Scaffold`	Show learning scaffold	returned.ssv
`NodeNet`	`NodeNet`	Show node network	returned.ssv

Scripts

Command	Usage	Effect
Include script	`<filename>.ssv`	Interpret another script inline
Autoexec	`autoexec.ssv` (in `Scripts/../`)	Runs automatically on startup

Usage Patterns

Simple run

help
exit

Training + Cogitation

Train
Cogitate test_input.ssv 1.0

Evaluation

eval "run1" 1.2

Autoexec example

set_depth 5
register_afferent 0 temp 5
register_efferent 0 heater
Train

Flags Logic

1. Write commands into `Control_Panel.ssv`
2. Set `Control_Panel_Flag.ssv` → `1`
3. GaiaOS runs, then sets `Control_Panel_Finished.ssv` → `1`
4. Read `Output/returned.ssv`

GaiaOS File Flow Diagram

Legend

1. Write your commands into `Control_Panel.ssv`.
2. Set `Control_Panel_Flag.ssv` = `1` to “ring the bell.”
3. GaiaOS notices, runs the commands.
4. When done, GaiaOS writes `1` into `Control_Panel_Finished.ssv`.
5. Read results in `Output/returned.ssv` or efferent files.
