GUI Backend Developer Guide

Overview

- The GUI backend exposes a TCP control and telemetry surface for the ground station.
- It listens on port `1029`, accepts up to eight concurrent clients, and uses newline-delimited ASCII commands.
- Cached state includes station metadata, RF configuration, antenna status, satellite telemetry, pass schedule, and recent events.

Architecture

- **Threaded TCP server**: `gui_backend_start()` launches `gui_backend_thread()`, which runs a blocking `select()` loop with a one-second timeout (`src/gui_backend.c:1165–1338`).
- **Client tracking**: each connection is represented by `struct gui_backend_client { fd, buffer[1024], size }` (`src/gui_backend.c:43`).
- **State cache**: `gui_state` (mutex guarded) stores the latest uplink/downlink summaries, rotator state, station information, RF settings, satellite snapshot, pass schedule, and a 64-entry event ring (`src/gui_backend.c:50–109`).
- **Telemetry broadcast**: once per second the backend sends a JSON telemetry snapshot to every connected client (`src/gui_backend.c:1207–1338`).
- **Event ring buffer**: `gui_backend_push_event()` timestamps, stores, and (when necessary) drops the oldest entries (`src/gui_backend.c:166–190`).

Subsystem Interfaces

Exported helpers in `src/gui_backend.h` keep the cache current:

Function	Purpose
`gui_backend_set_station_info(const char *name, double lat, double lon, double alt, double true_north)`	Records station identity, location, orientation, and update time (`src/gui_backend.c:272–318`).
`gui_backend_set_mode(gui_backend_mode_t mode)`	Updates station mode (`IDLE`, `TRACKING`, `MAINTENANCE`) and logs a GUI event (`src/gui_backend.c:304–314`).
`gui_backend_set_emergency_stop(int engaged)`	Toggles the emergency-stop latch and records an event (`src/gui_backend.c:319–341`).
`gui_backend_update_pass_schedule(const gui_backend_pass_t *passes, size_t count)`	Replaces the pass schedule (up to 16 entries) (`src/gui_backend.c:343–365`).
`gui_backend_update_satellite(const gui_backend_satellite_t *sat)`	Updates satellite telemetry fields and TLE epoch (`src/gui_backend.c:365–367`, `src/gui_backend.c:418–440`).

Function	Purpose
`gui_backend_notify_uplink/downlink`	Records the latest transport transaction and emits an event (`src/gui_backend.c:193-252`).
`gui_backend_notify_rotator(int az, int el, int success)`	Tracks rotator command results for the GUI (`src/gui_backend.c:254-270`).

Snapshot and Telemetry

- `gui_backend_snapshot()` captures cached state, computes TLE age, generates ISO timestamps, and renders the pass schedule as JSON (`src/gui_backend.c:418-639`).
- `gui_backend_send_status_payload()` emits JSON with the following shape (`src/gui_backend.c:640-687`):

```
{
  "type": "status" | "telemetry",
  "station": {
    "name": "...",
    "mode": "IDLE|TRACKING|MAINTENANCE",
    "emergency_stop": true|false,
    "lat": double,
    "lon": double,
    "alt_m": double,
    "true_north_deg": double,
    "time_utc": "YYYY-MM-DDTHH:MM:SSZ",
    "time_local": "YYYY-MM-DDTHH:MM:SS"
  },
  "antenna": {
    "az_deg": double,
    "el_deg": double,
    "last_command_success": true|false
  },
  "rf": {
    "tx_hz": int,
    "rx_hz": int
  },
  "satellite": {
    "norad": uint32,
    "lat_deg": double,
    "lon_deg": double,
    "alt_km": double,
    "velocity_km_s": double,
    "range_km": double,
    "range_rate_km_s": double,
    "tle_age_sec": long
  },
  "passes": [
    {
      "name": "...",
```

```
      "aos": "YYYY-MM-DDTHH:MM:SSZ",
      "los": "YYYY-MM-DDTHH:MM:SSZ",
      "duration_sec": uint16,
      "peak_elevation_deg": uint16
    }
  ],
  "faults": []
}
```

- `STATUS` replies include `OK STATUS` and `END` framing; the periodic broadcast omits the framing and sets `"type":"telemetry"`.

Command Protocol

Commands are ASCII lines terminated by `\n`. The parser is case-insensitive and validates argument counts (`src/gui_backend.c:1024-1108`).

Command	Behavior
`PING`	Responds `OK PONG`.
`HELP`	Prints available commands.
`STATUS`	Returns framed JSON snapshot.
`SET_MODE <idle|tracking|maintenance>`	Updates station mode; replies `OK SET_MODE value`.
`SET_EMERGENCY <true|false|1|0|on|off>`	Sets emergency stop; replies `OK SET_EMERGENCY true|false`.
`SET_SAT <1..255>`	Calls `mcs_sat_sel()`; on success returns `OK SATELLITE`.
`SET_TX <freq_hz>` / `SET_RX <freq_hz>`	Invokes Doppler setters, caches frequency, and acknowledges.
`SET_AZEL <az> <el>` (alias `SET_ROTATOR`)	Sends rotator command via `serial_set_az_el`.
`SEND_PACKET <pri> <src> <dst> <dst_port> <src_port> <hmac> <xtea> <rdp> <crc> <hex_payload>`	Converts payload to bytes, uses `send_packet_struct`, and replies `OK SEND_PACKET <len>`.
`GET_EVENTS [count]`	Streams the newest events as text lines: timestamp, severity, origin, summary, detail.
`LAST_UPLINK` / `LAST_DOWNLINK`	Prints the latest uplink/downlink summaries (origin, bytes, status, file path, node IDs).

Invalid or incomplete commands yield `ERROR` responses.

Event History

- `gui_backend_push_event()` stamps entries with `CLOCK_REALTIME` and stores them in the ring at `gui_state.events` (`src/gui_backend.c:166–190`).
- `gui_backend_event_type_t` is mapped to strings (`UPLINK`, `DOWNLINK`, `INFO`, `ERROR`) by `gui_backend_event_type_to_string()` (`src/gui_backend.c:1322–1374`).
- `GET_EVENTS` enumerates the newest entries, respecting optional limits, and formats them as plain text (`src/gui_backend.c:936–1000`).

Telemetry Broadcast Loop

1. `select()` waits with a one-second timeout (`struct timeval timeout = {1, 0};`).
2. After each wake, `gui_backend_emit_status_to_all()` sends the latest telemetry snapshot to every client (`src/gui_backend.c:689–705`).
3. Socket activity is processed by accepting new clients or reading buffered data; dead sockets are closed when `recv()` returns `<= 0`.

Integration Guidelines

Backend Contributors

- Use the exported setters (`gui_backend_set_station_info`, `gui_backend_update_satellite`, etc.) rather than mutating `gui_state` directly.
- Maintain thread safety by holding `gui_state.lock` inside helpers; avoid long-running operations while holding the mutex.
- When adding new telemetry fields, extend both `gui_state` and the snapshot serialization so JSON stays consistent.

Front-end Clients

- Open a long-lived TCP socket to port `1029`.
- Send commands as ASCII lines (e.g., `STATUS\n`); expect `OK`/`ERROR` or JSON responses.
- Consume the continuous `"telemetry"` JSON feed; use the `"type"` field to distinguish from `STATUS`.
- Poll `GET_EVENTS` to render historical activity; consider parsing severity strings to color-code entries.
- After issuing setters (`SET_MODE`, `SET_EMERGENCY`, etc.), rely on the telemetry feed to verify the state change.

Extensibility Notes

- `faults_json` is currently an empty array; populate it from the event ring or a dedicated fault queue to deliver push alerts.
- If more robust messaging is needed, you can replace the text protocol with JSON requests while retaining the existing sockets.
- Increase `GUI_BACKEND_MAX_CLIENTS` if more concurrent GUI tools are expected—be mindful of select()'s FD limits.
- To minimize broadcast size, consider batching or sending only changes (diff-based telemetry).

v1.0 - 20 OCT 2025