

protocol_tool.py

The command-line tool for moving `protocols-N.txt` files between the PC and the Generator, in both directions. Everything else here is built on it: the GUI calls it, and the editor imports it for reading and writing files.

What it does

Two modes:

- **download** — Generator → PC. Reads a `protocols-N.txt` off the device's SD card and saves it locally.
- **upload** — PC → Generator. Validates and reformats the local file first, then writes it to the device's SD card.

Upload is the direction that runs the correction pass, because that's the file a human or a spreadsheet program may have touched.

The file name matters

The file must be named exactly `protocols-N.txt`, where N is 0–99. The tool reads N out of the filename and tells the device which SD card slot to read or write. Any other name is rejected before the serial port is even opened.

`protocols-0.txt` is special: it's the live/default file. Uploading to slot 0 makes the device reload it into EEPROM immediately. Uploading to any other slot leaves it parked on the SD card until you activate it from screen 5's File= dial.

Using it

Download a file from the Generator:

```
python3 protocol_tool.py download --dir ~/Desktop --file protocols-0.txt
```

Upload an edited file back:

```
python3 protocol_tool.py upload --dir ~/Desktop --file protocols-3.txt
```

On a different serial port:

```
python3 protocol_tool.py upload --file protocols-0.txt --port /dev/ttyUSB1
```

Flag	Default	Meaning
<code>--file</code>	<i>required</i>	<code>protocols-N.txt, N</code>

Flag	Default	Meaning
		= 0–99
--dir	.	Folder the file lives in / is written to
--port	/dev/ttyUSB0	Serial port

Start the matching mode on the Generator first — **USB Comm – Send To CPU** for download, **USB Comm – Receive Fm CPU** for upload. The script then waits up to 20 seconds for the device’s handshake.

The validation pass (upload only)

Before sending anything, the file is checked and canonicalized:

- Strips a UTF-8 BOM if a spreadsheet added one.
- Skips blank lines, # comments, and header lines with no commas.
- Cleans stray quotes and whitespace from every field.
- **9-field lines** are frequency records — reformatted to 2 decimal places.
- **11-field lines** are the control record — reformatted as integers, plus the sweep increment to 2 decimals.
- Run times are bounds-checked: frequency time 0–9999 **seconds**, program time 0–9999 **minutes**. A raw millisecond value in either field is rejected rather than uploaded as an absurd run time.
- Requires at least one frequency record and **exactly one** control record.
- Rejects a file holding more than **275** frequency records — the Generator’s EEPROM image cannot hold more. Caught here rather than on the device, where it would not surface until the next boot.

If anything fails, *every* problem is reported at once with line numbers, and nothing is sent. The device is never left half-written.

If it fails

must be named 'protocols-N.txt' — rename the file. N tells the device which SD slot to use.

Timed out waiting for UPLOAD_READY / DOWNLOAD_READY — the Generator isn’t in the matching mode, or more than 20 seconds passed. Start the device side first.

Cannot upload -- file needs fixing — read the listed line numbers. Nothing was sent.

Serial error: ... Errno 16 / resource busy — close the Arduino IDE serial monitor or MiniCom.

Upload finished, but no UPLOAD_OK seen — the transfer ran but the device never confirmed. Check the RX: lines for an ERROR: message.

Related

- `protocol_tool_gui.py` — point-and-click front end that calls this script
- `protocol_editor.py` — grid editor for the files this script moves