

The protocols-N.txt File Format

Describes the protocol file format as implemented in the ESP32 Four Channel Generator firmware.

A protocol file is plain-text CSV. It tells the Generator which frequencies to produce, on which of its four channels, at what duty cycle, for how long, and in what order. Everything the device does in Protocol or Sweep mode comes out of one of these files.

This page is written to be complete on its own. An AI assistant given this page and nothing else has everything it needs to write a working file.

1. The file name is functional

The file must be named exactly `protocols-N.txt`, where N is 0–99. N is the SD card slot. The PC tools read N out of the name and tell the device which slot to write, so `my-protocol.txt` is rejected before the serial port is even opened.

`protocols-0.txt` is the **live/default** file. Writing to slot 0 makes the device reload it into EEPROM immediately. Writing to any other slot parks it on the SD card until it is activated from screen 5's File= dial.

2. Two kinds of record

A file is a list of **frequency records**, followed by exactly **one control record**, which must be the last line.

Record	Fields	How you recognize it
Frequency	9	first field is 1 or higher
Control	11	first field is 0

Also allowed anywhere, and ignored: blank lines, lines beginning with #, and a single header line with no commas in it (for example `Protocols-14 #`).

Maximum 275 frequency records. The device's EEPROM image cannot hold more.

3. The frequency record — 9 fields

Rec#, Freq1, Duty1, Freq2, Duty2, Freq3, Duty3, Freq4, Duty4

One record holds a complete setting for all four channels at once.

Field	Range	Notes
Rec#	1 and up	Must run consecutively from 1 with no gaps
Freq1-Freq4	0.04 – 65535.00 Hz	Hz, 2 decimal places
Duty1-Duty4	0.00 – 1.00	A fraction, not a percentage. 0.50 is 50%

The frequency floor is hardware: the ESP32 MCPWM cannot go below 160 MHz / $(256 \times 256 \times 65536) = 0.0373$ Hz, and the firmware clamps anything lower to 0.04. The ceiling is the 16-bit timer maximum.

To silence a channel you are not using, set its duty cycle to 0.00. The firmware computes $100 - (\text{Duty} \times 100)$, so a duty of 0.00 produces exactly the same value used to shut the outputs off, and the channel is held low. There is no “off” frequency — 0.00 in a frequency field is out of range.

Example — 7.83 Hz on channel 1, 40 Hz on channel 2, channels 3 and 4 silent:

1,7.83,0.50,40.00,0.50,40.00,0.00,40.00,0.00

4. The control record — 11 fields

Exactly one per file, always the last line, always starting with 0.

0, MemGrp, Mode, FreqTime, ProgTime, StartFreqGrp, StopFreqGrp, StartSweepGrp, StopSweepGrp, SweepInc, File#

#	Field	Range	Meaning
1	0	literal	Marks this as the control record
2	MemGrp	a real Rec#	The record the Generator starts on
3	Mode	1–4	See below
4	FreqTime	0 – 9999	Seconds each record runs. 0 = half a second
5	ProgTime	0 – 9999	Minutes the whole program runs. 0 = half a minute
6	StartFreqGrp	a real Rec#	First record of the protocol range
7	StopFreqGrp	a real Rec#	Last record of the

#	Field	Range	Meaning
			protocol range
8	StartSweepGrp	a real Rec#	Record holding the sweep start frequency
9	StopSweepGrp	a real Rec#	Record holding the sweep limit frequency
10	SweepInc	0.00 – 9999.99	Hz added per step in sweep mode
11	File#	0 – 99	Must match the N in the filename

The four modes

Mode 1 — Single. Holds one record's four frequencies and does not advance. StartFreqGrp/StopFreqGrp are not used.

Mode 2 — Protocol. This is the stepping mode, and the one most programs want. The device plays record StartFreqGrp, waits FreqTime seconds, plays the next record, and so on through StopFreqGrp — then **wraps back to StartFreqGrp and repeats**. It keeps repeating until ProgTime minutes have elapsed, then stops.

Mode 3 — Sweep. Channel 1 is swept. It starts at the StartSweepGrp record's Freq1, adds SweepInc every FreqTime seconds, and stops climbing at the StopSweepGrp record's Freq1 — landing exactly on the limit rather than overshooting. Channels 2, 3 and 4 keep the StartSweepGrp record's values throughout. When the sweep reaches the limit it restarts from the beginning, and continues until ProgTime expires.

Mode 4 — Adjust. A live frequency control. The Generator starts from the record showing on Setup=0 — after start-up, the MemGrp record — and while it runs, turning the knob moves Channel 1's frequency up or down in steps of SweepInc Hz, or 1 Hz if SweepInc is 0. The black button switches the knob between frequency and duty cycle. Channels 2, 3 and 4 hold that record's values, and nothing is written back to the file.

5. Five rules that are easy to get wrong

1. FreqTime is in seconds, ProgTime is in minutes. They are different units in the same record. Three minutes per frequency is 180. One hour total is 60.

2. Repeating is automatic — there is no repeat count. In modes 2 and 3 the range loops on its own until ProgTime runs out. Do not duplicate records to make a program run longer, and do not look for a “cycles” field. There isn't one.

3. 0000 in either time field means *half*, not zero. A FreqTime of 0 is half a second. A ProgTime of 0 is half a minute. This is deliberate, it matches what the front panel has

always done when either field is dialed to 0000, and it is what the device itself writes to the card when you save a half-value. It is not a way to say “no limit” — for a long run, use a large number.

4. All four group fields must name a record that exists — including StartSweepGrp and StopSweepGrp in mode 2, where the sweep is never used. They are read regardless. If your file has 3 records, 1 and 3 are safe values.

5. File# must match the filename. protocols-30.txt needs 30 in field 11.

6. A complete worked example

The request: channel 1 plays 7.83 Hz, then 432 Hz, then 528 Hz. Channel 2 holds 40 Hz throughout. Each frequency runs 3 minutes. The program loops for one hour.

The file — protocols-30.txt:

```
Protocols-30 #
1,7.83,0.50,40.00,0.50,40.00,0.00,40.00,0.00
2,432.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
3,528.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
0,1,2,180,60,1,3,1,3,1.00,30
```

Reading the control record: mode 2 steps records; 180 seconds is 3 minutes per frequency; 60 minutes is the hour; 1, 3 is the range to walk and then repeat; 1, 3 again satisfies the sweep fields, which mode 2 ignores; 1.00 is an unused sweep increment; 30 matches the filename.

Channels 3 and 4 carry 40.00 at duty 0.00, which holds them silent.

A timing check worth doing. 60 minutes ÷ 180 seconds = exactly 20 frequency slots, and 20 ÷ 3 records = 6 full passes with 2 slots left over. The hour ends part-way through the seventh pass, so 7.83 and 432 each play 7 times and 528 plays 6. For whole cycles, use 54 minutes (6 passes) or 63 minutes (7 passes).

7. Adding a program to a file that already has one

You do not need a new file. A file holds up to 275 records, so a new program can be appended to the spare space in an existing one and selected by pointing the control record at it.

Starting from a protocols-8.txt whose records 1–15 are already in use, append the three new records as 16, 17, 18 and retarget the range:

```
16,7.83,0.50,40.00,0.50,40.00,0.00,40.00,0.00
17,432.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
18,528.00,0.50,40.00,0.50,40.00,0.00,40.00,0.00
0,1,2,180,60,16,18,3,4,1.00,8
```

Records 1–15 are untouched and simply are not visited. Changing 16, 18 back to the old values restores the original program.

8. Checking your work before you plug anything in

The PC tools include the device's own validator, so a file can be verified offline:

```
python3 -c "import protocol_tool;  
protocol_tool.load_and_correct('protocols-30.txt')"
```

Silence means the file is good. Any problem is reported with a line number, and *every* problem is listed at once. See `protocol_tool_guide`.

The validator's 0–9999 range on the two time fields is correct as written — 0 is a legal value meaning half, per rule 3.

9. Getting the file onto the Generator

```
python3 protocol_tool.py upload --dir ~/Desktop --file protocols-30.txt
```

Start the device side first — USB Comm → *Receive From CPU* — then run the command. The PC waits 20 seconds for the handshake.

To pull an existing file off the device and edit it instead, use *Send To CPU* on the device and download on the PC.

Related

- `protocol_tool_guide` — moving files to and from the device
- `protocol_editor_guide` — grid editor for these files
- `protocol_tool_gui_guide` — point-and-click transfers