

S5000 Lite Scale Protocol

How an app talks to the S5000 Lite kitchen scale over Bluetooth Low Energy: how to find the scale, which packets it sends, how to compute the weight from them, and which commands control it.

REFERENCE IMPLEMENTATION S5000/lib/s5000_protocol.dart	TRANSPORT BLE GATT, service FFF0	DOCUMENT DATE 5 Oct 2026
---	-------------------------------------	-----------------------------

01

Verification status

The protocol comes from the manufacturer's documentation and was implemented in the S5000 test app. Not everything has been tried on a real scale, so every command in this document carries one of three labels.

- Verified on device

● Observed differently from the docs

● Per documentation, unverified

02

Finding the scale

During a scan the scale can be recognised by either of two things in its advertisement. One match is enough.

- The service list contains UUID FFF0 , in full 0000fff0-0000-1000-8000-00805f9b34fb .
- The manufacturer data (AD type 0xFF) starts with the bytes FA FB .

Manufacturer data

0 FA	1 FB	2 VV	3 TT	4... ..
header	header	version	device type	more data, MAC

TYPE (BYTE 3)	DEVICE
00	personal scale
01	body fat scale
02	kitchen scale, which is what the S5000 Lite is
03	blood pressure monitor

Watch out for BLE libraries. Most BLE libraries (for example flutter_blue_plus or Android ScanRecord) treat the first two bytes of the manufacturer data as a company ID and return them separately as a little-endian number. You then get FA FB as the key 0xFBFA and the rest of the packet as the value. The reference app rebuilds the full packet like this: [key & 0xFF, key >> 8, ...value] .

03

GATT

UUID	TYPE	PROPERTY	DIRECTION
FFF0	service		
FFF1	characteristic	notify	scale → app
FFF2	characteristic	write	app → scale

- The full UUID is 0000fff1-0000-1000-8000-00805f9b34fb , and likewise for fff0 and fff2 .
- After connecting, enable notifications on FFF1 by writing 01 00 to the CCCD descriptor 0x2902 . Most libraries do this with a single call.
- The reference app writes to FFF2 with response when the characteristic supports it, otherwise without response.
- Every packet fits into the default MTU (20 bytes of payload), so no MTU change is needed.

04

Packet format

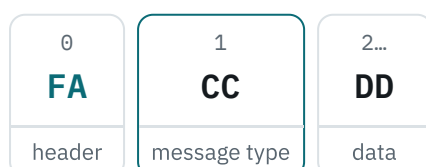
Every packet in both directions starts with the header 0xFA . In Java and Kotlin this value shows up as the signed byte -6 . The second byte is the command or message code. Packets carry no checksum.

App → scale



The documentation says the data is followed by 0x00 padding. The reference app sends no padding and the scale still responds. Commands without data (tare, battery, history) also work as two-byte packets without a length byte, which is verified for tare.

Scale → app



The codes overlap between the two directions but mean different things. For example, `FA 04` sent to the scale asks for the number of history records, while `FA 04` received from the scale is a live weight. Always keep track of which side a packet came from.

05

Weight packet

The scale streams the weight with message `0x04` for as long as the load is changing. Once the value settles it sends message `0x05`. Both share the same layout. According to the reference implementation, the history record `0x06` uses the same layout too.

0 FA	1 04	2 SS	3 UU	4 DD	5 HI	6 LO	7... 00
header	04 / 05 / 06	status	unit	decimals	value, high	value, low	zeros on kitchen scale

```
value = ((HI << 8) | LO) / 10^DD
if SS == 2: value = -value
```

The value is an unsigned 16-bit big-endian number. The sign is carried by the status byte. With one decimal place the largest value that can be shown is therefore 6553.5 units.

STATUS (SS)	MEANING
00	OK
01	overload
02	negative value, typically after tare when something is lifted off the scale
03	low battery
04	zero search, the scale is zeroing itself

UNIT (UU)	MEANING	NOTE
00	kg	
01	lb	
02	st:lb	compound unit, see below
03	g	usual kitchen-scale unit
04	ml	
05	oz	
06	lb:oz	compound unit, see below

The same unit codes apply to the set-unit command. For the compound units `02` and `06`, the documentation says the high nibble of byte DD holds the decimals of the first unit and the low nibble those of the second. The reference app uses only the low nibble. This has not been verified on a device.

Examples

PACKET	RESULT
FA 04 00 03 01 01 F4	live 50.0 g 0x01F4 = 500, one decimal place
FA 05 00 04 00 02 58	stable 600 ml 0x0258 = 600, no decimal places
FA 04 02 03 00 00 1E	live -30 g status 02 = negative value

06

Messages from the scale

All messages arrive as notifications on `FFF1`.

CODE	MESSAGE	FORMAT
00-03	Acknowledgement of commands 00 to 03 00 user, 01 time, 02 unit, 03 tare	FA CC SS SS: 00 success, 01 error
04	Live weight	FA 04 SS UU DD HI LO
05	Stable weight	FA 05 SS UU DD HI LO ...
06	History record	FA 06 SS UU DD HI LO ...
07	History record count	FA 07 NN
08	Battery level	FA 08 PP PP = percent, 0x55 = 85 %
09	Blood pressure data	other device type only, the scale never sends it
0A	Calibration response	FA 0A FF FF = calibration phase

07

Commands to the scale

All commands are written to `FFF2`.

CODE	COMMAND	PACKET	RESPONSE	STATUS
03	Tare (zero)	FA 03	FA 03 00	● Verified 2 bytes without a length byte are enough
02	Set unit codes as in the unit table	FA 02 01 UU grams: FA 02 01 03	FA 02 SS	● Docs

CODE	COMMAND	PACKET	RESPONSE	STATUS
06	Battery level	FA 06	FA 08 PP	● Observed the S5000 Lite apparently does not answer
01	Time sync unix time in seconds, big-endian	FA 01 04 T3 T2 T1 T0	FA 01 SS	● Docs
00	Set user UU user number, XX sex (00 female, 01 male), AA age, HH height in cm	FA 00 04 UU XX AA HH FA 00 04 01 01 1E AF	FA 00 SS	● Docs for body scales, the kitchen scale most likely ignores it
04	History record count	FA 04	FA 07 NN	● Docs
05	All history records	FA 05	n × FA 06 ...	● Docs
07	Display LED brightness MM: 00 off, 01 gradual, 02 bright	FA 07 01 MM	not specified	● Docs
08	Calibration MM: 00 show internal code, 01 start calibration	FA 08 01 MM	FA 0A FF	● Docs

Calibration can throw the scale off. The command FA 08 01 01 puts the scale into calibration mode. Without the correct calibration weight the scale may lose accuracy. Do not expose it in an end-user app.

08

Communication flow

1. Scan and find a device with service FFF0 or manufacturer data FA FB .
2. Connect and discover services and characteristics.
3. Enable notifications on FFF1 .
4. Optionally send a time sync and set the unit, for example FA 02 01 03 for grams.
5. Receive 0x04 messages with the live weight and 0x05 once the value settles.
6. To zero the scale, send FA 03 . The scale answers FA 03 00 .

Practical notes

Auto power-off

The protocol has no command to disable the scale's auto power-off. The reference app keeps the scale awake by sending a harmless command roughly every 20 seconds. By default it re-sends the current unit, that is `FA 02 01 UU`. A battery query is not enough, because the S5000 Lite apparently does not answer it. If the unit command were not enough either, the app can also tare an empty scale or toggle between g and ml, which forces the firmware to redraw the display.

Reconnecting

When the scale turns off, the connection drops. On Android, calling `connect` with `autoConnect = true` works well. The system then waits passively and reconnects as soon as someone turns the scale on, with no retry loops or timeouts.

Android permissions

- Android 12 and newer: `BLUETOOTH_SCAN` and `BLUETOOTH_CONNECT`.
- Android 11 and older: `BLUETOOTH`, `BLUETOOTH_ADMIN` and `ACCESS_FINE_LOCATION`.

Code samples

Dart (from the reference app)

shortened

```
const int header = 0xFA;

// Commands to FFF2
List<int> tare() => const [header, 0x03];
List<int> setUnit(int unit) => [header, 0x02, 0x01, unit & 0xFF];
List<int> setTime(DateTime t) {
  final ts = t.millisecondsSinceEpoch ~/ 1000;
  return [header, 0x01, 0x04,
    (ts >> 24) & 0xFF, (ts >> 16) & 0xFF, (ts >> 8) & 0xFF, ts & 0xFF];
}

// Weight packet from FFF1 (message 0x04 or 0x05)
double? parseWeight(List<int> d) {
  if (d.length < 7 || d[0] != header) return null;
  if (d[1] != 0x04 && d[1] != 0x05) return null;
  final status = d[2];
  final decimals = (d[3] == 0x02 || d[3] == 0x06) ? d[4] & 0x0F : d[4];
  var value = ((d[5] << 8) | d[6]) / math.pow(10, decimals);
  if (status == 2) value = -value;
  return value;
}
```

Python with bleak

illustrative, not run against a scale

```
import asyncio
from bleak import BleakClient, BleakScanner
```

```

NOTIFY = "0000fff1-0000-1000-8000-00805f9b34fb"
WRITE  = "0000fff2-0000-1000-8000-00805f9b34fb"
UNITS  = {0: "kg", 1: "lb", 2: "st:lb", 3: "g", 4: "ml", 5: "oz", 6: "lb:oz"}

def on_data(_, data: bytearray):
    if len(data) >= 7 and data[0] == 0xFA and data[1] in (0x04, 0x05):
        dec = data[4] & 0x0F if data[3] in (2, 6) else data[4]
        value = ((data[5] << 8) | data[6]) / 10 ** dec
        if data[2] == 2:
            value = -value
        state = "stable" if data[1] == 0x05 else "live"
        print(f"{value:.{dec}f} {UNITS.get(data[3], '?')} ({state})")

async def main():
    dev = await BleakScanner.find_device_by_filter(
        lambda d, adv: "0000fff0-0000-1000-8000-00805f9b34fb" in adv.service_uuids)
    async with BleakClient(dev) as client:
        await client.start_notify(NOTIFY, on_data)
        await client.write_gatt_char(WRITE, bytes([0xFA, 0x03])) # tare
        await asyncio.sleep(60)

asyncio.run(main())

```

11

Open questions

The documentation is unclear on these points, or they have not been tried on a device.

Anyone who verifies them should add the result here.

- Decimal places for the compound units st:lb and lb:oz, that is, whether byte DD really splits into two nibbles.
- The exact format of history record `0x06` and whether it includes the measurement time.
- Whether the scale requires `0x00` padding after command data, and up to what length.
- The byte order of the time in command `0x01`. The reference app sends big-endian.
- Whether the S5000 Lite answers the battery query under other conditions, for example only after a time sync.
- What the scale sends back for the LED and calibration commands.

Compiled from the S5000 project: files `lib/s5000_protocol.dart` and `lib/scale_session.dart`.