



Tech Bulletin: POS Shows “!\$...\$” and Junk Instead of Caller ID

Why it happens, why it's new, and the one-line fix — in the receiving software, not the unit

Field & Integration Tech Bulletin — 30JUN26 · Applies to: Whozz Calling, Vertex, Vertex Solace

Team — A customer's POS started showing garbage in the caller-ID field — things like `!$...$` followed by stray text such as `time.google.com` — on what should have been a clean call. I want to be clear up front: **the CallerID unit is fine and was set up correctly.** This is a bug in the **software that receives our data.** It is hunting for a `$` sign to find where the caller data starts, and this particular unit's **serial number happens to contain \$ characters**, so the software latches onto the wrong spot and reads junk. The fix is one small change on the receiving side, and it is the method we already document in our Ethernet Link Supplement: **skip the fixed header and start reading after the 21st character — never search for the first \$.**

1. What you see on the POS

Instead of a clean caller name and number, the caller-ID field shows the unit's serial fragment and leftover bytes — for example:

```
# What the POS displays (wrong):
Phone: WIRELESS CALLERNN90time.google.-
      ...or a bare line that begins with !$...$ and trails off into raw text.

# What it should display:
Phone: 770-555-1234   Name: WIRELESS CALLER
```

The clean part of the name is usually right; the trouble is the extra characters jammed onto it, and the occasional line that is *all* junk.

2. Why it happens — the “\$” is the serial number, not the data

Every call-event packet we put on the wire starts with a fixed-length header that carries the **unit number** and **serial number**, and only then comes the actual call record. To save space, those numbers are compressed from 12 digits down to 6, and that compression turns certain digit pairs into punctuation. **The digit pair 24 compresses to a \$ (dollar sign), and the pair 21 compresses to a !.**

Look at the serial on the unit that broke: **21-240115-16-24** . It contains **two 24 pairs and a 21** — so inside the header that serial literally reads **! \$... \$** . That is exactly the **!\$...\$** you are seeing on the POS. **It is not corruption. It is the serial number doing its normal thing.**

```
# Every call packet is laid out like this – the header is ALWAYS 21 characters:
^^<U>nnnnnn<S>nnnnnn$01 I S 0000 R 1 06/30 10:45 AM 770-555-1234 WIRELESS CALLER
|_____ 21-character header _____|▲ the real call record starts here (character 22)
                                   the ONE real "$" separator

# The problem: this serial puts EXTRA "$" signs INSIDE the header...
^^<U>nnnnnn<S>!...$...$01 I S 0000 ...
                ▲ software that "finds the first $" stops HERE – inside the serial – and
                reads junk
```

3. Why this is new today (and where it will show up next)

This trap has always existed, but it only springs when a unit's **serial or unit number contains a 24 pair**. Most serials don't, so "find the first **\$** " worked by luck for years and nobody noticed. The unit at this site is simply the first one in the field whose serial contains **24** — twice. **Any future unit with a 24 in its serial will do the same thing on this POS until the receiving software is fixed.** That is why I want this fixed properly in the software, not worked around one unit at a time.

This is documented behavior, not a defect. Our **Ethernet Link Supplement** spells it out: *"when an uncompressed '24' is in a serial number or unit number... a \$ is produced when compressed... a good method... is to start reading the Caller ID data after the 21st character."*

4. The fix — in the receiving (POS / CRM) software

Two small rules in whatever code reads our UDP packets. Both are easy, and together they end this for good:

Rule 1 — Skip the header by length; never search for "\$". The header is **always 21 characters**. Start reading the call record at **character 22** (i.e. after the 21st character) and ignore everything before it. Do *not* scan the packet for the first **\$** — that is the line that breaks, because the serial can contain **\$** . In code that counts from zero, that's simply: drop the first 21 bytes of the UDP *payload* — e.g. `record = payload.slice(21)` — working on the datagram's data, not the whole captured packet.

Rule 2 — Only accept real call packets; drop the rest. Read only **UDP packets on port 3520** that begin with our identifier **^^** , are **at least 21 characters** long, and whose record fields look like a real call (a line number, an **I / O** direction, an **S / E / B** status). Discard everything else — that includes our own setup and acknowledgement packets (they

start with `^^Id / ^^ok`) and unrelated LAN chatter (the `time.google.com` -type text from other devices). This keeps anything that isn't a genuine call record out of the caller-ID field.

With both rules in place, the serial's `$` signs are never mistaken for the data, the fields line up, and non-call traffic can't leak in. This works for every unit we've ever shipped and every one we build next — the header is always 21 characters and the format never changes.

5. What this is NOT — don't chase the wrong fix

It is not a unit setting. The unit at the site was in normal configuration — sending only at the start of the call, with Detailed Data, Extra SIP Fields, and the rest all *off*. None of those toggles cause this and none of them fix it. Don't go changing Data Output options chasing this.

Do not use “Ignore Leading Characters in Caller ID Name Field.” That option trims characters off the front of the caller's *name* (it's meant to drop a phone-company prefix). Turning it up would erase real names — “WIRELESS CALLER” is only 15 characters — and it still wouldn't remove the junk, because the junk isn't in the name field at all. Wrong lever.

One-line summary for the POS developer: read UDP/3520 payloads that start with `^^` and are at least 21 characters, skip the first 21 characters, and parse the fixed-width call record from there (`payload.slice(21)`) — do not search for the first `$` , because the serial number can contain `$` .