JTDX CE3TSK fork – Improvements for WW Digi 2026

The FT8 decoder and the contest support: all enhancements and fixes

Tihomir Sokcevic CE3TSK

September 6, 2026

Part I — The FT8 decoder

FT8 decoder - all enhancements and fixes

The decoder itself - how it works, these changes in context, the three new approaches and the presets - is described in `FT8_DECODER.pdf`.

The full campaign is documented step by step in the project notes; this is the inventory. Everything is measured on the benchmark capture `wav/full_band_16.wav` (104-message validated reference set; WSJT-X 3.0.2's best mode decodes 79 of it).

The scripts, recorded audio and expectation files named below (`test/decode/...`, `test/harness/...`, `test/fortran/...`, `wav/...`) are the author's measurement harness. It is not part of the published source repository; it will be made available for download at ce3tsk.com. The paths are kept so the entries stay verifiable once it is.

Decoding improvements (more messages)

1. **Candidate cap raised 450 → 2000** in `sync8` - the old cap silently dropped real candidates on a busy band (step 5).
2. **Second near-zero-DT sync peak per bin** - a second DT peak is kept when the main peak passes the threshold (syncmin2 = 1.1 near NFQSO), with a squared thinning weight on pass 2, so overlapping stations on the same frequency both get a shot (step 5).
3. `-O` **deep OSD** - OSD order 2 for every candidate as a selectable mode (step 5), later part of the WeakSignals preset.
4. **Second slicing pass** (`-z`) - after pass 1, all threads' subtractions are merged onto the original audio and the slices re-run offset by half a slice; recovers slice-boundary and under-stronger signals, never below the single-thread set (step 8).
5. **Alternate-approach pass** (`-X`) - a full re-decode with the opposite recipe (SWL off, 7 cycles, OSD order 2) on the merged-subtraction band; breaks the 7-of-17-weakest ceiling every single recipe hit (step 10).
6. **Decode range decoupled from the waterfall width** - the decoder gets the full 0-5000 Hz regardless of the displayed waterfall (step 7); since 2026-08-25 a *decode bandwidth* submenu: the waterfall's width, twelve fixed ranges (100-3300, 100-3200 ... 200-2400, 300-3300, 50-3650 Hz) or the full band, "testing only" (`decoderange.h`).

7. **Presets** (steps 6/10) - Default (3 cycles: 71 msgs, 0.4 s), MaxDecodes (SWL-6 + second slicing: 81, 3.3 s), WeakSignals (5 cycles + OSD + second slicing: 79, 2.1 s), Exhaustive (SWL-5 + alternate pass: **85 msgs** / **8 of the 17 weakest, 2.6 s** at 12 threads), Ensemble (exhaustive + 3 members: **98 msgs, about 8 s**, see item 21); thread-aware, shown as Custom when the manual controls match none.

Correctness and determinism fixes

Threaded results used to vary 65-81 from run to run; after these, every run is byte-identical.

8. **Shared audio buffer** `dd8` made threadprivate with copyin - threads subtracted decoded signals from the buffer other threads were still reading (step 8, the big one).
9. `ft8_downsample` **C++ cache** race fixed.
10. `osd174_91` `first_osd` **init race** - double-checked, with a flush before publishing.
11. `four2a` **FFTW plan-cache publish race** - an entry was published (`nplan` incremented) before its plan existed; now slot filled $\rightarrow$ plan created $\rightarrow$ flush $\rightarrow$ publish, with a re-check and the NPMAX check under the lock. Found from a GUI replay that gave 83 where the CLI gave 85 forty times in a row.
12. `gen_ft8wave` **tables** made threadprivate.
13. **Decoded-message store** - the callback counter moved into a critical section with a bounds check; messages could be lost or overwrite at the array end.
14. **Documented latent trap** (not changed): FFTW patience 2 (MEASURE) plans in place on the live data and destroys it (41 messages). The GUI uses patience 1.

Machine-independence (step 11)

15. **Fixed 12-slice band grid decoupled from the thread count** - threads pick slices up dynamically (`schedule(dynamic,1)`); per-slice state is indexed by slice, subtraction deltas merge in slice order. Result: **the same message set at every thread count $\geq$ 2** - a 2-core and a 16-core machine decode identically, `-j` only changes the time. `JTDX_SLICES` overrides the grid (capped at 24); a narrow band gets fewer slices automatically.

Speed

Exhaustive preset at 12 threads: 2.6 s (7.7 s at 2 threads); single-thread SWL-6 14.6 $\rightarrow$ 6.8 s.

16. **Long-FFT plan cache** made threadprivate (step 5).
17. `sync8` **window-sum tabulation** (step 12) - the 17-bin (7-bin strided in the AGC variant) window sum was recomputed for every DT lag although it depends only on the bin and the symbol; tabulated once with the identical expression, bit for bit exact, the lag loops halved.
18. **OSD bit-packing rewrite** (step 13) - GF(2) rows packed into 3 x int64 words: word-wise elimination, incremental codewords by row XOR, soft distances by masked sums (bit-exact FP order preserved), and the 2 MB per-call hash memset replaced by a 2^ntau direct-keyed table with O(1) tail append. Proven identical to the byte reference on 39,000+ randomized cases.

Test infrastructure built alongside

19. **File-decode mode** (`jt9files` in `src/lib/jt9.f90`) - the campaign's instrument: decode any wav from the CLI with every GUI setting as an option (`-8 -W -K -C -O -z -X -j -L -H ...`), including `-Q` AGC compensation; `JTDX_DUMP_PARAMS` and `JTDX_MEMO_STATS` diagnostics.
20. **Regression and equivalence suite** - 43-row deterministic decode regression (`test/decode/regress.sh` presets, thread counts, 16/32 bit, AGC metric, slice overrides, previous-period feedback, ensemble, pipeline, every message printed once), the pipeline's abort/budget/restore check (`bgabort.sh`), OSD equivalence tests (3,000 fixed-seed + 20,000-case seedable extended run), 21 unit tests - all green.

Ensemble decoding (2026-08-25, the project notes)

21. **Ensemble members** (`src/lib/ft8ensemble.f90`, `decoder.f90`, `-M n`, preset "ensemble") - what the old non-deterministic runs were sampling, made reproducible. The chain's threshold decisions fall differently when the input moves by less than the noise (a 1 % dither swings one decode between 77 and 86), so the pristine band is decoded again through fixed perturbations - a 128-sample delay, a 3 % seeded dither, -0.8 / +1.2 Hz shifts - each with its own recipe, sharing the duplicate arrays so only new messages print; a member's finds are never stored as the previous period's signal, and reported DT / frequency are corrected for the perturbation. Members gated by the thread count (3 from 12, 2 from 6, 1 from 3): **98 messages in 7.9 s** at 12 threads, 95 in 5.7-6.4 s at 6-8, 92 in 6.6-7.9 s at 3-4; `-M 5` reaches 101. Deterministic, the same set at 3 and 12 threads, every gained message in the validated reference set, the sparse capture unharmed; with the previous-period feedback 100. The *ensemble effort* level (off / auto / 1-5, `FT8EnsembleEffort`, `-M`) overrides the gate for testing and for faster or slower machines.

Pipeline ensemble (2026-08-25, the project notes)

22. **Pipeline ensemble** (`ft8_unit` / `ft8_background` in `decoder.f90`, `jt9a.f90`, `-B n`, preset "pipeline ensemble", *background effort* menu) - the decoder process used to idle ~12 of every 15 s, TX or RX periods alike, and the reply decision is taken ~1 s after the decode starts, so everything slower than that was already "late". Now a fast RX phase (SWL-5, 78 messages at 1.2 s) prints `<DecodeFinished>` as before, and a background phase fills the idle time with more units on the retained band - the deferred alternate pass, the ensemble members, a residual pass (every known decode subtracted, then a lowered threshold), extra members - each preceded by a check of the GUI's `.lock` (removed = next decode wanted, the unit stops within a pass) and, in auto, of the time budget. Only new messages print, with the period's UTC; a `<BackgroundFinished>` line carries the count. **78 + 24 = 102** on the capture with every unit (about 12 s of units fit at 12 threads), deterministic, the same set at 3 and 12 threads. The instrumented retry unit answered the ensemble plan's open question: the RX phase's LDPC failures are noise candidates (2438 of 2475 fail the sync gates again), so the members' gains come from the candidate/subtraction stage - the residual pass is the unit that finds what the members miss. The TX background has its own complete recipe (Decode → FT8 decoding → expert → *TX background*, the RX controls mirrored: SWL, cycles, sensitivities, OSD, passes, ensemble effort); presets set both phases. Preset "pipeline run" (3 cycles in the period, so the reply is decided before TX keys) reaches **71 + 22 = 93**; "pipeline ensemble" (SWL-5 in the period) **78 + 24 = 102**.

Recipes on a realistic range (2026-08-26, the project notes)

23. **100-3100 Hz decodes as the full band does inside it.** A narrower decode range used to lose decodes (default 71 → 61, exhaustive 85 → 80) for two decoder reasons: the sync baseline (`sync8`'s 40th percentile) was taken over the decoded range, where a busy band puts it on signals, and the 12-slice grid was laid over the range, making 250 Hz slices that stop subtraction at their edges. The baseline now uses a fixed 0-5000 Hz span and the grid is anchored to the full band (417 Hz cells from 0 Hz, a range gets the cells it intersects): 100-3100 Hz gives default 70, SWL-5 77, exhaustive 84 - the full band minus the one message at 78 Hz - with the full band itself unchanged to the message. A fresh configuration now starts at 100-3100 Hz (`DECODE_BW_DEFAULT`); stored choices and the old full-band boolean keep their meaning.

24. **The subtraction window follows each unit's SWL flag.** `cwfilter` built `subtractft8`'s smoothing window once per decode from the period's SWL setting, so after a plain-cycles RX phase every SWL member and the SWL background recipe subtracted with the wrong window and the pipeline stopped 9 messages short (91 against 100). Per-unit now: "pipeline run" 93 → 99 on the full band; "5 cycles, sensitivity 2, one member" 87 messages with 10 of the 17 weakest in 2.4 s on 100-3100 Hz - the in-budget winner of the recipe sweep (140 runs), which also found sensitivity 2 to be the cheapest gain in the menu (+6 for 0.2 s) and OSD order 2 no weak-signal lever on a crowded band.

25. **Presets on the measured recipes, and the 27 s background window (P7).** Presets are now Default (3 cycles), **Max efficiency** (5 cycles + sensitivity 2: 76 / 7 weakest in ~0.75 s, the knee of the decodes-per-second curve), **Max decodes** (5 cycles + sensitivity 2 + 1 member from 8 threads: 87 / 10 weakest in 2.4 s - it retires "weakest signals" and "exhaustive", both dominated), Ensemble (unchanged, no time budget), **Pipeline ensemble** (SWL-4 + 1 member in the period, 86 on the table, then 100 - the ceiling of the range, the background trimmed to the units that ever found anything, 16.5 s) and **Pipeline run** (5 cycles + sensitivity 2, then 97). And the GUI no longer decodes its own TX period while the TX background is enabled (`src/decodebudget.h`, `test_budget`): the decode before our TX hands the decoder a two-period window (`params.nbgbudget`), so the background runs through our 15 s transmission and the other station's 12.64 s, about 27 s, and the whole unit list completes every period instead of stopping at member 4. A band or mode change aborts a running background through a sentinel file, so the previous band's audio is never printed under the new dial (stock JTDX has the same hole at the width of one decode; the pipeline would have stretched it to ~26 s). Every preset's two phases with times are tabled in the project notes and `test/decode/preset_data.tsv` (pipeline ensemble: 86 in 2.8 s, then +14 in 16.3 s).

26. **RX "budget auto"** (P8, the project notes) - a new RX ensemble effort mode: after the base recipe, members run while the next one's learned cost fits the ~2.7 s reply budget (`FT8RXBudget`), each run seeding the next member's estimate, so a quiet band gets its spare reply time spent on the members that measurably pay there (the sparse capture: 7 messages at reply time instead of 6, 3 of the members run from the first period) while a crowded band stays at the base recipe plus one member (87, the Max decodes result). Menu: RX → ensemble effort → "budget auto"; presets unchanged. The background's effort auto has been budget-driven since the pipeline.

27. **The classic unit in the background** (P9, the project notes). A side-by-side run against the Sep-2025 build on air (202 common periods) showed the old plain 6/9-cycle decoder still printing a few decodes the pipeline never got. The dominant pattern (`CQ CT7AUT IM67` at -24 dB, 30 periods) turned out to be JTDX's DX-call AP search on a station the operator had clicked in the old instance - both decoders decode it in file mode with that DX call and

grid set - not decoder strength. What remained real was measured by replaying all 207 saved periods through one process per recipe: the pipeline finds 270 messages the 6/9-cycle decoder does not, the 6-cycle decoder 11 the pipeline does not (all -22..-24 dB, in 11 periods; 9 cycles adds 4). So the pipeline background gained a cheap second unit, the *classic unit*: the plain non-SWL 6-cycle decode on the pristine band (`FT8BgClassicCycles`, file mode `-n`, unit letter `c`), 0.3-0.5 s on air, in every preset's background. Inside the pipeline it measures +6 in 207 periods (5 real, one false decode), nothing lost; the standalone decoder's own chain of finds belongs to it being the RX phase and stays a follow-up. On air afterwards (08:00-14:02 UTC, 1323 common periods) the P9 build printed 9201 decodes to the old decoder's 8540: 722 real finds only it made against 84 only the old one made, net +661 (+7.7 %). Different seeds would not have helped - the plain decoder is deterministic; the dither members already run. Three diagnostic hooks came with the analysis (`JTDX_CAND_TRACE`, `JTDX_STOPHINT`, `JTDX_DXCSEARCH`) and a minimal FT8 file mode for the Sep-2025 source, so the old decoder can be run on saved periods (scratch build, not part of the tree).

28. **Pipeline max decodes light** (P10, the project notes) - a lightweight pipeline preset for the 15 s cycle with TX: Max decodes' RX phase (87 at 2.4 s, the reply decided there) and a fixed ~6 s TX background - the plain 6-cycle pass first (the classic decode, 0.5 s), ensemble members 2 and 3, the residual pass - for 94 in all on the benchmark, 7 on the sparse capture. No budget auto, no cap: the list was measured once (after this RX phase the SWL recipe finds nothing in 2.7 s, member 2 finds 3 in 1.1 s, member 3 finds 3 in 2.9 s, the residual 2 in 1.7 s) and is expressed entirely through the existing TX background controls, so no decoder or params change. On air (16:57-17:57 UTC, 240 periods, same audio) it printed 5739 decodes to 5100 for the Sep-2025 decoder at 9 cycles + sensitivity 2: 674 real finds only it made against 36, more in 212 periods against 2, net +639 (+12.5 %); 273 of its decodes came from the ~3-6 s background.

29. **Fix: the background's abort re-ran the decode in a loop** (found on air 2026-08-27). `jt9a.f90` jumped back to the decode after any aborted background, assuming the GUI had removed `.lock` for the next period. A `.bgabort` abort (band or mode change) leaves `.lock` in place, so with Monitor off - no decode() coming to clear the sentinel - the decoder re-decoded the same retained audio again and again, every message printed once more per pass (mostly as hint decodes: two periods in the production ALL.TXT with 16 and 49 messages repeated up to 18 times). Now the jump happens only when `.lock` is gone; a sentinel abort falls through to the normal wait. GUI-path only - the file mode was never affected, so it is not covered by `bgabort.sh`; confirmed on air the same day (Monitor off, then a band change: no repeats).

30. **Every preset on one hour of air** (the project notes; rerun 2026-09-05 on the deployed pair with the real stock rows added - item 82 has the refreshed figures). *That hour, the FT4 one of item 48 and the two crowded-band files are published as 16-bit WAV, with a cross-platform script that reproduces these rows on any machine:* **<https://ce3tsk.com/download/wav/>**. The 240 periods of the side by side replayed through each preset with the previous-period feedback carried, the classical JTDX recipe (9 cycles, sensitivity 2, JTDX's one-period hint memory) as the reference at 5264 decodes in 0.8 s per period. With the production decoder (P11's four-period hint memory): Max efficiency 5427 (+3.1 %, 0.4 s); Max decodes 5687 (+8.0 %, 1.1 s); *Pipeline max decodes light* 6093 (+15.7 %, 1.1 s + 4.0 s background); Ensemble 5987 (+13.7 %, 4.6 s); Pipeline ensemble 6220 (+18.2 %, 1.2 + 11.0 s); Pipeline ensemble full 6227 (+18.3 %, 1.2 + 12.2 s); Pipeline run 6239 (+18.5 %, 0.4 + 12.7 s). The light preset takes 87 % of the pipeline ensemble's gain at 42 % of its time and is worse than classical in 2 periods of 240. WSJT-X 3.0.2 on the same hour with its own ini settings (the multithreaded FT8 decoder, 9 passes, sensitivity 3, AP): 5188 (-1.4

%, 1.3 s); its standard FT8 decoder at depth 3: 4782 (-9.2 %, 2.5 s); WSJT-X improved 3.2.0 with the same ini settings: 5173 (-1.7 %, 1.2 s). The full pipelines reach 96 % of the union of all eleven sets (6491), the classical recipe 81 %. (Before P11 the JTDX rows measured 5119 / 5357 / 5750 / 5674 / 5888 / 5905 / 5904 - archived in `results-hint1/`.) By SNR, the gains of every JTDX recipe over the classical decoder are about one fifth at -24 dB or below, one half at -23..-21 dB and one quarter at -20..-18 dB - the light preset more than doubles the classical count at $\leq$ -24 dB (152 $\rightarrow$ 346) and adds 54 % at -23..-21 (677 $\rightarrow$ 1046) - while nothing changes above -17 dB; the WSJT-X decoders lose a quarter of the classical decodes at $\leq$ -24 dB. Recommendation from it: Max decodes for the best bang for the buck (+8 % for 1.1 s), Pipeline max decodes light for the best medium effort (+16 % for 5.1 s, 4 s of it in the idle half of the cycle - production), Pipeline ensemble for the best result (+18 % for 12.2 s; run/full within 20 messages, chosen by their RX phase). The presets submenu marks them (bold, a coloured dot, the tier and its figures in front of the name: orange "best value", green "best medium effort", blue "best results" on Pipeline ensemble, purple "most results" on run, red "max effort" on full, and grey "best power" on Max efficiency; the light preset is additionally underlined and labelled "recommended:", Pipeline ensemble underlined too - the least CPU for +3 %).

31. **The hint decoder remembers four periods** (P11, the project notes). JTDX's ^ hint decoder tried a stored message only if it was decoded in the previous same-parity period, so one missed period broke a station's chain (the `CT7AUT` case). The message lists of the same parity now reach four periods back (two minutes; the older lists searched only when the newer hold nothing, a hint decode retiring its entry where it came from; `JTDX_HINT_DEPTH=1..8` overrides, 1 = JTDX). On the one-hour on-air set: classical 9 cycles 5264 $\rightarrow$ 5575 (+5.9 %), the light preset 5750 $\rightarrow$ 6095 (+6.0 %) - more than any depth-1 pipeline - at no measurable time; 86-89 % of the gains are chains continuing through a missed slot, none of them nonstandard text. The cost: 1-2 stale hint decodes an hour (the previous message on a re-used frequency when the new one shares most of its bits), which is why the memory stops at two minutes although eight periods still measured +7 %. `bgabort.sh` case `hintdepth`. With the longer memory a band or mode change now also empties the hint message lists (all depths) and the call/DT lists, which JTDX kept across the change - they are keyed by audio frequency and DT and mean nothing under another dial (case `bandchange`, file-mode hook `JTDX_BANDCHANGE=n`).

32. **The presets set their environment, and the expert submenu** (P12, the project notes). Applying any preset now also switches *early start of decoder* off (every measurement here is on the whole period's audio, and the pipeline timing assumes it) and *wideband DX Call search* on (the DX-call AP search over the whole range - the section 11 finding); `PRESET_EARLY_START`, `PRESET_WIDE_DXCALL_SEARCH`, the decode trigger re-derived at once, neither part of the preset identification. Decode $\rightarrow$ FT8 decoding is *Presets* and one *expert* submenu with everything else (RX, TX background, early start, wideband DX Call search, decode bandwidth).

33. **The preset lamp** (P13, the project notes). Under the Contest lamp, above Tune: "Preset X" with one letter per preset in the submenu's order (3 P V R E B M O; "Custom" otherwise), lit in the preset's dot colour for the six marked ones, plain text in a grey box for the others, greyed outside FT8; follows every recipe control, the thread count, the mode and the presets themselves.

34. **Report-range gate against false decodes** (2026-08-28). A false decode of the a-priori kind - your call fixed by AP, the remaining 48 bits random from an OSD trial that happens to pass the 14-bit CRC - prints as `CE3TSK JW1GPY/R 216`: a random second call, the random rover bit, and a g15 field in the report region that no transmitter can produce (`irpt` above 105, i.e. beyond report -31 after the -101 wrap). `unpack77` now rejects `irpt > 105` in

type 1/2 messages, as it already did for out-of-range contest exchanges; two such lines in the month's ALL.TXT, both at -22 dB or less, both `/R`. Pinned in `test/fortran/pack_roundtrip` (32505 accepted, 32506 and 32752 rejected).

35. **RTTY Roundup exchanges switched off** (2026-08-28). Type 3 messages (`W9XYZ K1ABC R 579 MA`) are not worked in FT8/FT4 on the air any more; the ~10 a month in ALL.TXT (`<...> RU5MRW R 569 7493`, all at -19 dB and below, singleton calls) were false decodes. `unpack77` rejects the type behind the source-level constant `LRTTYRU_DECODE` (module `packjt77`, default `.false.`, nothing in the GUI or ini; packing for TX is untouched). Likewise type 0.5 telemetry (`1564C7B008B8AD4BAC`, ~50 a month, nearly all below -20 dB - nobody sends FT8 telemetry on HF) behind `LTELEMETRY_DECODE`. The WW Digi exchange `R FF46` is type 1 and untouched (pinned by the same test and a simulated decode). Both pinned in `pack_roundtrip`.

36. **The ALLCALL7 lookup switched off** (2026-08-28). `chkflscall` rejected a message whose callsigns are all absent from `ALLCALL7.TXT`; the file is the July 2024 community update, so every QSO between two stations licensed since was thrown away whatever the SNR. Measured on the 240 on-air periods with the same decoder, file present vs absent: classical 5226 vs 5267 (44 lost, ~3 of them false), light 6053 vs 6095 (53 lost, ~4 false) - `RI1FJL KN6JIB DM13`, `ER35MD N7EYE DM33` at -13 dB among the losses. Source constant `LALLCALL7_FILTER=.false.` in `chkflscall.f90` (one place, all six call sites and the DX-call search); the file stays where it is. The 1-hour table had been measured without the file all along (`run_presets.sh` never copied it), so production now decodes as the table says; the regression sets, generated with the file, gained the rejected messages (`test/decode/onair_1h/results-allcall7/`, `allcall7_check.sh`).

Net result

85 of the 104 reference messages in 2.6 s (exhaustive), **98 in 7.9 s** (ensemble) and **102 with the pipeline ensemble** (78 on the table for the reply decision at 1.2 s, the rest in the decoder's idle time), deterministic, identical on any machine at 2 threads or more - vs 79 for WSJT-X 3.0.2's best mode. On the realistic 100-3100 Hz range (2026-08-26): **87 in 2.4 s** with 5 cycles, sensitivity 2 and one member (preset *max decodes*), and **100 - every message any recipe finds on that range - with the SWL-4 + member pipeline** (preset *pipeline ensemble*, its background now running through our own TX period).

37. **FT4 hint memory** (2026-08-29) - JTDX's FT4 decoder had no hint pass at all (`stophint` there only trims the AP passes). Now every decoded message is kept with its frequency and DT for the last four same-parity 7.5 s periods (`ft4_mod1.f90`: `ft4cur`, `ft4even/ft4odd(:,k)`, rotated at the start of each period by `decoder.f90`, emptied on a band or mode change) and, once every ordinary pass has failed on a candidate in all three DT segments, a stored message within 3 Hz / 0.1 s is tried once more as one extra pass with all 77 message bits fixed - the mechanism of the RR73 a-priori type 6, so the acceptance criterion is JTDX's own; a second sweep over the segments, gated on a stored message within 20 Hz, so a hint never pre-empts an ordinary decode. Hint decodes carry the `^` marker like FT8's and retire the entry they used. Measured on 73 consecutive periods of the WW Digi FT4 traffic (14.080 MHz, 14:00-14:17 UTC, `wav/files/ft4_1400/`): 513 → 537 messages (+4.7 %), 27 hint decodes, no (period, message) lost, +0.7 s of CPU over the 73 periods; every hinted message had been decoded at that frequency in an earlier period. On a seeded `ft4sim` chain at -17 dB 7 of 8 periods decode without and 8 of 8 with the memory; at -18 dB 3 → 5. `JTDX_FT4_HINT_DEPTH=0..8` (default 4, 0 = JTDX's behaviour); `test/decode/ft4hint.sh`.

38. **File mode made reproducible** (2026-08-29, found with valgrind on the item above) - the

file-mode driver (jt9.f90) handed the decoder a params block built on its own stack with the callsign options left as they were when not given: hiscall was stack garbage, so the "MyCall DxCall" a-priori passes decoded differently from one run to the next (four decodes of calls to CE3TSK came and went between runs of the same binary). The option strings are blanked before parsing and the block is zeroed before it is filled. Live decoding was never affected - the GUI's block starts zeroed. A blank mycall (no -c) is now honestly "no station configured", which the FT8 decoder treats differently from any call, so the regression rows pass -c CE3TSK -G FF46 by default: with a callsign the recorded sets reproduce exactly (the old garbage had acted as some nonblank call). JTDX_SIM_SEED=n makes the simulators reproducible for the same reason (init_random_seed.c).

39. **FT4 candidate cap** (2026-08-29) - getcandidates4 stopped at its 100th spectral peak in *frequency* order, so a band with more peaks lost the top of the passband outright. Now a module variable (ft4_mod1.f90, default 400, arrays 1000, JTDX_FT4_MAXCAND overrides). Measured on the WW Digi FT4 hour: 41-97 candidates a period (median 70) against 180-300 for an FT8 hour - an FT4 signal is ~83 Hz wide against FT8's ~50 Hz, so fewer fit and the cap binds far less often; it never did in this hour (identical output at 100, 200, 400 and 1000, same time), but 97 is one busy period from the cliff. The FT4 period's closing line now reports <ncand> like FT8's.

40. **FT4 honours unpack77's verdict** (2026-08-29) - the FT4 path accepted any decode whose text was non-empty, so the report-range gate and the RTTY-Roundup / telemetry switches (items 33-35, packjt77.f90) never applied to it: on the WSJT-X FT4 sample TU; F09NJF WZ8DWC 559 7995 at -20 dB got through, on the contest hour CE3TSK QU1RK CE34. Now if(.not.unpk77_success) cycle, exactly as ft8b does. On that sample this also drops the ten RTTY-Roundup exchanges it was recorded for, which is the switch's stated policy; flip LRTTYRU_DECODE for an RTTY contest.

41. **FT4 decode range anchored** (2026-08-29) - the *decode bandwidth* submenu applied to FT4 since August, but getcandidates4 fitted its noise baseline over the decoded range only, so a narrower range changed the candidate threshold at its edges - the effect the FT8 fix (item 6, nfawide/nfbwide) removed. The FT4 baseline is now fitted over ft4_baseline's own fixed span (200-4910 Hz) whatever the range; the peak search stays inside the range. On the 73-period FT4 hour: before, 100-3300 Hz against the full band differed by 12 missing / 11 extra messages inside 100-3300 Hz; after, the two sets are identical, and the full-band output is unchanged line for line.

42. **FT4 on the packed OSD** (2026-08-29) - ft4_decode called its own stock osd4_174_91, a copy of the FT8 OSD that differed only in thread plumbing, so the 64-bit bit-packed rewrite (item 18) never reached FT4. It now calls osd174_91 (thread slot 1). Output bit-identical on the 73-period FT4 hour (537 lines); OSD time 18.9 → 5.0 s for the same 22 452 calls (0.84 → 0.22 ms), the whole decode 43.6 → 31.0 s (-29 %) - measured with JTDX_FT4_TIMING while production was running, so absolute seconds are upper bounds. The remaining FT4 profile: sync search ~58 %, OSD ~16 %, BP ~16 %.

43. **FT4 sync twiddle hoisted** (2026-08-29) - sync4d multiplied the four Costas templates by the frequency twiddle on every call, though the twiddle is constant across the whole DT sweep of a frequency step (~130 starts). The templates now live in ft4_sync_mod (sync4d.f90), the twiddled copies are formed once per frequency step (ft4_sync_twiddle) and sync4d_tw runs sync4d's own sums on them - the same products, so the sync values and the decode output are bit-identical (537 lines on the 73-period hour). Sync search 17.9 → 11.6 s (0.34 → 0.22 us per call, 53 M calls); with item 42 the FT4 decode went 43.6 → 24.5 s (-44 %), loaded-machine figures. sync4d itself is unchanged and still used nowhere else.

44. **FT4 "more passes" - a negative result** (2026-08-29). With the FT4 decode 44 % faster (items 42-43) the obvious question was whether to spend the time on deeper decoding. Experiment hooks (`JTDX_FT4_OSDDEEP`, `_SYNCQUAL`, `_NSP`, `_BPITER`, `_IDFSTP`, `_SYNCMIN`; defaults unchanged) over the 73-period FT4 hour: OSD depth 4 costs 2.1x for +3/-3 messages, depth 5 costs 5.7x (2 s a period, unusable for a reply) for +14/-3; the sync-quality gate at 18 or 16, four subtraction passes, 80 BP iterations, a 2 Hz coarse frequency step and sync thresholds 1.1 / 1.0 all return the same 537 messages for 3-20 % more time. JTDX's FT4 recipe already extracts what its candidate/sync design can find; further gains have to come from a different search (the alternate pass on the residual and the ensemble of the FT8 work, and the same-frequency subtraction weakness seen on the WSJT-X sample), not from more of the same - post-contest work. The freed time stays as margin for the reply.

45. **FT4 alternate pass and ensemble member - measured, not adopted** (2026-08-29). The two FT8 approaches transplanted behind hooks: `JTDX_FT4_ALT=1` runs one more subtraction pass over the residual with the other DT search windows (the SWL/plain swap), `JTDX_FT4_DITHER=1` a third sweep that re-runs the bit metrics and passes on the candidate with FT8's deterministic 3 % dither. On the 73-period hour: alternate +2 messages for +64 % time, dither +1 for +98 %, both +3 (540) for 3.2x, nothing lost - against +3-4 % each in FT8. FT4 already decodes three bit-metric variants per candidate (a small ensemble of its own) and its shorter symbols leave fewer marginal candidates. One of the three is a station calling us (`CE3TSK UA3RLE R LO02`), so the idea is not worthless - it would need an FT4 TX background to be free, which is the post-contest FT4 plan's threading item. Both are in the menu as Decode → FT4 decoding → expert (`FT4AltPass`, `FT4Dither` in the ini, two new fields at the end of the parameter block - jtdx and jtdxjt9 from the same build only), off by default; the env hooks remain for file mode.

46. **FT4 gets FT8's false-decode gate** (2026-08-29). `ft8b` sends every doubtful decode through `chkfalse8` (callsign/grid plausibility, `chkgrid`, the *R* and *73* rules, `callsign_q`): doubtful means weaker than -20.5 dB, or reached through an AP type 1-3 (CQ / mycall / mycall+hiscall masks), unless the message carries our own or the QSO partner's call. `ft4_decode` had none of this - unpack77's verdict (item 40) was its only filter. The same gate is now in the FT4 path with the threshold at -17.5 dB (FT4 is 3.5 dB less sensitive than FT8, so this is the same "bottom of the range" the FT8 value marks), env hook `JTDX_FT4_FALSEGATE=<dB>` (-30..30; 30 = check everything). Measured on the 73-period hour: gates -17.5, -14 and -10 reject nothing (537 lines stay 537); gate 30 rejects exactly one line, the -9 dB `CQ WWVE5MX` of 14:16:30 - a type-4 (nonstandard call) decode that is a mangled variant of a real `CQ WW VE5MX` and would have been printed as a station. So on real traffic the FT4 decoder's own sync/CRC/OSD chain and item 40 leave next to nothing for the gate to catch; it is there for the same reason as FT8's, the rare ghost at the threshold, and costs nothing (one `chkfalse8` per weak decode). `JTDX_FT4_TIMING=1` prints each rejection (`ft4 reject i3 .. n3 .. iaptype .. snr .. <message>`). Pinned in `test/decode/ft4hint.sh`: gate 30 rejects 1 on the hour, the default rejects 0; and on simulated ghosts (`CQ K1ABC PM95`, a US call in a Japanese grid, `ft4sim` seed 7 at -18 dB, gate raised to -10 through the hook because the sim reports -16..-18): 8 decodes with the gate off, 0 with it (4 rejections - the other four were hint-chain decodes that never start once the seed is rejected); the same ghost at -10 dB passes the default gate; `CQ CE3TSK PM95` is rejected without a station and passed with `-c CE3TSK` (the own-call exemption). Review found one divergence from ft8b and fixed it: the hashed-first-call flag (`<...> CALL ...`) that makes `chkfalse8` also check the second call's grid was passed as false; it is now `message(1:1).eq.'<'` as in ft8b. Seen in passing, shared with FT8 since it is the same `chkfalse8`: a `CQ WW CALL GRID` line is not grid-checked (`CQ WW UA9ABC FF46` passes), nor is the second call of `CALL1 CALL2 GRID`.

47. **FT8:** `ft8b` **read a never-set array in its sub-pass 3** (2026-08-29, valgrind). Asked whether the FT8 decoder had uninitialised-memory problems like the file-mode driver (item 38), a memcheck pass over one on-air period first produced thousands of reports - nearly all of them the valgrind artefact for gfortran's multi-MB stack frames (`ft8b`, `sync8`, `cwfilter`: "client switching stacks", cured by `-max-stackframe`). The clean run left 178 contexts, all with one origin: `ft8b`'s local `cscs(0:7,79)` (the symbol spectra of the unreversed signal) is filled only on the reversed passes (`lreverse`, passes 2/5/7), yet sub-pass 3/6/9 - run for every candidate that looks like a CQ or a call to us, on every pass - forms its bit metrics from `|cscs|^2 + |csr|^2`. On passes 1/3/4/6 it therefore combined the candidate's reversed spectra with whatever the stack held, in practice the `cscs` of the last candidate of a reversed pass: a sub-pass that could only ever decode by accident. Everything else valgrind flagged (the metric maxima at 848-871, `bpdecode174_91`, `osd174_91`, `indexx`, line 1471) was that garbage propagating. Fix: `cscs = cs` on the non-reversed passes (`ft8b.f90`, one line), which makes sub-pass 3 the same |normal|^2 + |reversed|^2 combination it is on the reversed ones. Also `xsnr` is now initialised before the `ft8b` call in `ft8_decode.f90` (it was left unset on a failed candidate and `nint`-ed; harmless, `nsnr` is only used after a CRC pass). Result: the 44 regression rows are unchanged. One-hour on-air comparison, light preset, 6 threads on the loaded contest machine: old engine 6092 decodes, fixed engine 6092; 231 of 240 periods identical, 6 messages lost and 6 gained, all between -15 and -26 dB - the noise the light preset's time budgets produce between any two runs under load. Neutral: the sub-pass never decoded anything on those passes with the garbage, and with real data it adds nothing the other sub-passes miss. The fix is still right - a decoder must not read undefined memory, and the garbage was whatever the previous candidate left, i.e. non-reproducible between machines and runs. The deterministic classical preset (9 cycles, sensitivity 2, no time budgets) on the same hour separates effect from noise: old engine 5570 decodes, fixed 5574; 230 of 240 periods identical, 4 lost and 8 gained (-15..-26 dB) - the sub-pass now decodes a little on those passes, and the changed subtraction order costs a few elsewhere. Net +4 (+0.07 %), i.e. the fix is safe and marginally positive. Single-period replays of the differing periods give identical message sets on both engines (and the 44 regression rows are unchanged): the differences only appear across the sequence, through the previous-period feedback and the hint lists, so there is no single-file row to pin; the classical hour above is the record. The guard is the new `test/decode/memcheck.sh` (one simulated FT8 period with a CQ - what drives sub-pass 3 - and one FT4 period under memcheck with the right `-max-stackframe`; any uninitialised-value context in `src/lib` fails), and it bites: with the one line reverted it fails with 96 decoder contexts (`ft8b.f90:871`, `bpdecode174_91`, ...), with the fix both periods are clean. Ready as `jtdxjt9` 4c7c66dd (jtdx and the parameter block unchanged); the FT4 gate went on air first, this one waits for the word.

48. **The FT4 options measured on a proper hour** (2026-08-30). Items 37, 44 and 45 were measured on 73 periods of 20 m traffic - too small, and recorded while listening rather than operating. The WW Digi night gave a better sample: `wav/files/ft4_night/` → `~/dev/wav_save/ft4_night`, **240 continuous receive periods, 2026-08-29 23:38:45 - 00:38:15 UTC on 14.080 MHz**, one hour of real operation and the same size as the FT8 `onair_1h` set, and published beside it at <https://ce3tsk.com/download/wav/>. Only every other 7.5 s period is in it - the station transmits on the other alternation - so it is one parity, exactly the sequence the live decoder sees, with the hint chain running over consecutive same-parity periods. Decoded at `-d 3`, 3 threads, with the contest still running (times are upper bounds):

```
Two references matter, because **the default is not JTDX's FT4**: it already carries the
depth-4 hint memory (item 37), which stock JTDX has no equivalent of. Percentages are given
against both, and the time against the default's 70 s.
```

```
| run | messages | against the default | against JTDX's FT4 | time |
|---|---|---|---|---|
| `JTDX_FT4_HINT_DEPTH=0` - JTDX's own FT4 | 1525 | -191, **-11.1 %** (gained 5, lost 196) | - | 64 s, 0.91x |
| **default, hint depth 4** | **1716** | - | +191, **+12.5 %** | 70 s, 1.00x |
| `JTDX_FT4_ALT=1` | 1733 | +17, +1.0 % (gained 18, lost 1) | +208, +13.6 % | 107 s, 1.53x |
| `JTDX_FT4_DITHER=1` | 1725 | +9, +0.5 % (gained 10, lost 1) | +200, +13.1 % | 135 s, 1.93x |
| both | 1744 | +28, +1.6 % (gained 29, lost 1) | +219, +14.4 % | 210 s, 3.00x |
| `JTDX_FT4_HINT_DEPTH=8` | 1732 | +16, +0.9 % (gained 25, lost 9) | +207, +13.6 % | 71 s, 1.01x |
| `JTDX_FT4_MAXCAND=800` | 1716 | 0 | +191, +12.5 % | 70 s, 1.00x |

What it changes:

- **The hint memory is worth far more than the small set showed**: +191 messages, +12.5 %,
  where the 73-period set gave +4.7 %. 192 of the 1716 decodes carry the `^` marker. Dense,
  repetitive contest traffic is exactly what a message memory feeds on.
- **Item 45's conclusion was too pessimistic.** Alternate pass and dither together now give
  +29 (+1.7 %) rather than +3 (+0.6 %), and the cost has to be read in absolute terms: 210 s
  for 240 periods is **0.87 s per period** against a 7.5 s period, at three threads, on a
  machine also running the contest. The reply deadline is nowhere near threatened, so "it
  would need an FT4 TX background to be free" was the wrong test - the budget already fits.
  Both stay off by default (nobody has flown them on air yet), but they are worth switching
  on in Decode -> FT4 decoding -> expert.
- **Depth 8 is not an improvement over 4**: +25 but -9, i.e. it starts resurrecting stale
  messages. Depth 4 stays the default.
- **The candidate cap still does not bind**, even in this traffic: 800 gives exactly the
  output 400 gives.
- **The false-decode gate (item 46) is safe**: at `JTDX_FT4_FALSEGATE=30`, which checks every
  decode, the output is identical to the default's, and only five duplicate a-priori
  re-decodes are rejected - messages already found by another pass. Real RR73s addressed to
  us survive at every gate value.
```

49. FT4 gets the whole ensemble, not one member of it (2026-08-30). Item 45 transplanted FT8's *dither* into FT4 and judged the ensemble idea on it. That was one member of three: `ft8ensemble.f90`'s `ens_perturb` perturbs a failed candidate by a **delay**, by additive **dither** and by a **frequency shift**. All three now run on the aligned FT4 candidate as members of `JTDX_FT4_ENSEMBLE=N` - member 1 is the old dither (N=1 is byte-identical to the previous run, so nothing measured before moves), 2 and 3 shift by +-0.6 Hz, 4 and 5 delay by +-1 sample, 6 draws different noise. A delay moves the reported DT with it and a frequency shift moves the reported frequency, so a member's decode is reported where the signal really was; the shift uses the phasor recurrence of `ft8ensemble`, so it is bit-identical on every machine. The hook is `JTDX_FT4_ENSEMBLE=N`, named as FT8 names its own member count (`FT8EnsembleEffort`, `actionFT8Ensemble1..5`); `JTDX_FT4_DITHER` was the name while the dither was the only member and still works. Measured on the 240-period hour of item 48, `-d 3`, 3 threads, contest running:

```
The default it is measured against already has the depth-4 hint memory, which JTDX's own
FT4 does not - the last column keeps that in view. Time is against the default's 70 s.

| run | messages | against the default | against JTDX's FT4 | time | per period |
|---|---|---|---|---|---|
| JTDX's own FT4 (hint 0) | 1525 | -191, -11.1 % | - | 64 s, 0.91x | 0.27 s |
| **default** | **1716** | - | +191, +12.5 % | 70 s, 1.00x | 0.29 s |
| 1 member - the old dither | 1725 | +9, +0.5 % | +200, +13.1 % | 135 s, 1.93x | 0.56 s |
| 2 members | 1763 | +47, +2.7 % (52 / -5) | +238, +15.6 % | 199 s, 2.84x | 0.83 s |
| 3 members | 1784 | +68, +4.0 % (81 / -13) | +259, +17.0 % | 258 s, 3.69x | 1.08 s |
| 4 members | 1797 | +81, +4.7 % (93 / -12) | +272, +17.8 % | 316 s, 4.51x | 1.32 s |
| **6 members** | **1815** | **+99, +5.8 %** (111 / -12) | **+290, +19.0 %** | 445 s, 6.36x | 1.85 s |
| **6 members + alternate pass** | **1832** | **+116, +6.8 %** (126 / -10) | **+307, +20.1 %** | 690 s, 9.86x |

These are the corrected figures. The first run of this table had the frequency members
shifting twice - the downsample centre moved *and* a phasor applied - so they probed +-1.2 Hz
while reporting +-0.6 Hz, which put a 0.6 Hz error on every decode they made. Fixed to the
```

downsample centre alone, which is exact because ‘f1‘ is what the report and ‘subtractft4‘
both use. Worth noting for later: the buggy +-1.2 Hz probe scored *better* (1834 and 1854),
so the offset the members use is itself a knob worth measuring.

Re-measured threaded, 2026-08-31 (deployed build ‘jtdxjt9 1c57c0f2‘, 12 threads, idle
machine, same 240-period hour). The table above was taken at 3 threads with the contest
running and before the item 52 determinism fixes; on the current build the whole ladder moves:

run	messages	vs stock FT4	ms/period	share of the 1360 ms deadline
stock FT4 (‘JTDX_FT4_HINT_DEPTH=0‘)	1533	-	87	6.4 %
default (hint depth 4)	1715	+182, +11.9 %	94	6.9 %
6 members	1816	+283, +18.5 %	535	39.3 %
6 members + alternate pass	**1829**	***+296, +19.3 %**	**840**	**61.8 %**

A real stock binary, 2026-09-01. The ‘JTDX_FT4_HINT_DEPTH=0‘ row above is *our* decoder
with the hint memory off, not stock, and it flattered us. JTDX v2.2.159 was built from the
pristine checkout at ‘/home/tata/dev/jtdx_analysis/jtdx‘ (HEAD ‘2a0e2bea‘, ‘Versions.cmake‘
‘WSJTX_VERSION_SUB 159‘) with **one** change: upstream ‘lib/jt9.f90‘ has no file-decoding
path at all - ‘use readwav‘, the wav header and the usage text are commented out and file
arguments fall straight through to the exit label - so ‘jt9files‘ was ported in. No decoding
code was touched. Stock runs single-threaded; it needs the rc159-era ALLCALL7 list, because a
current one overruns ‘call0c(11500)‘ in ‘jt65_mod9.f90‘ and aborts (the fork raised those
bounds tenfold and switched the lookup off).

run	messages	vs real stock	ms/period	threads
stock JTDX v2.2.159	**1341**	-	303	1
this decoder, hint memory off	1533	+192, +14.3 %	88	12
default (hint depth 4)	1715	+374, +27.9 %	94	12
3 members (recommended)	1787	+446, +33.3 %	313	12
5 members (most results)	1813	+472, +35.2 %	450	12
6 members + alternate pass	**1829**	***+488, +36.4 %**	840	12

So the headline against a genuine stock build is ***+36.4 %**, not the +19.3 % the stand-in
gave: the stand-in still carried every other FT4 fix, worth +192 on its own. The three parts
are now separable - decoder fixes +192, hint memory +182, ensemble and alternate pass +114.
It is not a superset: stock finds 31 messages the top row does not (46 against the
default recipe), which is worth saying whenever the +36.4 % is quoted.

Two things this settles. **The alternate pass does fit once FT4 is threaded**: 840 ms against
the 1360 ms deadline, where the 3-thread run needed 2.88 s - 212 % of the budget, late in
every period. And **the headline against stock is +19.3 %, not the +20.1 % this item first
reported**: the stock baseline itself rose from 1525 to 1533, because the item 52 fixes
recover decodes with the hint memory switched off too. Quote +19.3 % - it is the figure whose
baseline and top row come from the same build and the same thread count. The default and
6-member rows reproduce item 51’s re-measure (1715 and 1815/1816), which is what makes the
two new rows comparable to it.

***+99 messages, +5.8 %, and +116 (+6.8 %) with the alternate pass** - against the +9 the
single dither member gave. Measured from JTDX’s own FT4 the whole stack - hint memory,
ensemble and alternate pass together - is ***+307 messages, +20.1 %**, and it is worth saying
which part carries it: the hint memory +191 of those, the ensemble +116. The kinds are not equal: the two freque
the +111, the delays +30, the dither members +22. It is not the sync grid they are beating - a finer
coarse frequency step, which would be far cheaper, does nothing at all (‘JTDX_FT4_IDFSTP=2‘
+16/-12, ‘=1‘ +12/-12, DECODER_IMPROVEMENTS item 44 said the same on the small set). The
sync search already finds these candidates; what a member changes is the demodulation of one
it had given up on. The hint memory compounds it: hinted decodes rise from 192 to 233,
because a member’s decode is stored and re-tried in later periods.

What it costs, corrected: **the budget is not the 7.5 s period**. FT4 transmits 105 symbols of
576 samples (5.04 s) and the decoder is handed ‘NMAX‘ = 73728 samples = 6.144 s of it, so it
has **1.36 s** before the next period starts - the figure every "affordable against 7.5 s"
line in items 45 and 48 was measured against, wrongly. At six members the hour cost 1.85 s a

period on three threads under contest load, i.e. it would have overrun; the live decoder has twelve threads on sixteen cores, which is what has to be measured before a member count goes on air. If six does not fit, the answer is FT8's multithreading carried across unchanged rather than a second scheme - after the contest.
The 12 losses are repeats of two stations (‘CQ WW KN4USA EM95‘ five times, ‘W3UCA‘ twice) displaced by the changed subtraction order, all of them decoded in other periods anyway; three of the gains carry our own call, the ones that decide a QSO. Still off by default and file-mode only above N=1 - the menu switch means one member, and giving FT4 an FT8-style Off/1..6 submenu with i on air yet. That is the next step, together with deciding whether N=6 belongs in a preset.

50. The FT4 runtime table, taken properly (2026-08-30, after the contest, idle laptop with the browser closed, load 0.70). Items 48 and 49 were measured at three threads while the contest ran, and every "affordable" judgement in them was made against the wrong deadline - the 7.5 s period rather than the **1360 ms** the decoder actually has (105 symbols x 576 samples = 5.04 s transmitted, `NMAX` 73728 = 6.144 s captured, of 7.5 s). Redone on the 240-period hour, `-d 3`:

run	messages	ms/period	share of the 1360 ms deadline
JTDX's own FT4 (hint 0)	1525	257	19 %
default	**1716**	**284**	**21 %**
alternate pass alone	1733	436	32 %
1 member	1725	548	40 %
2 members	1763	789	58 %
3 members	1784	1026	75 %
alternate pass + 2 members	1785	1226	90 %
4 members	1797	1261	93 %
alternate pass + 3 members	1807	1588	**117 %**
6 members	1815	1776	**131 %**
6 members + alternate pass	1832	2760	**203 %**

FT4 is single threaded. *(Superseded on 2026-08-30 by item 51, which gave it FT8's slice threading. Everything from here to the end of this item describes the decoder as it was that morning; the conclusions it draws - six members unusable, three the ceiling - no longer hold. The measurement itself stands and is why the threading was worth doing.)* The control settles it: the default co 284 ms at twelve; three members 1025 ms and 1026 ms. ‘-j‘ changes nothing, because the slice fan-out and ‘numthreads‘ exist only in the FT8 path (‘decoder.f90‘ ~1027). So the contest-load figures were never distorted by load - they were already the true costs - and the deadline, not the machine, is what binds.

What it decides:

- **Six members cannot be used today**: 1776 ms against 1360 ms, so the reply would be late. Taking it on air, as was planned that night, would have failed.
- **Three members is the ceiling** at 75 %, with real headroom. Four is 93 %, which leaves nothing for a slower machine, so it is a menu entry rather than a preset.
- **The alternate pass is dominated.** With two members it costs 90 % of the budget for 1785 messages; three members alone give 1784 for 75 %. It stays in the menu - alone it is the cheapest gain there is, +17 messages for 11 points of budget - but no preset uses it.
- **This is what the threading work buys** (the project notes): at even fourfold efficiency six members would fall to ~450 ms, 33 %, and the whole ensemble becomes usable.

The presets follow the measurement: default 0 members, "recommended" 2, "most results" 3, none with the alternate pass; the menu labels carry each entry's share of the budget, as the FT8 ensemble menu carries its own numbers.

51. FT4 gets FT8's slice threading (2026-08-30, the project notes). Item 50 ended by saying six members needed threading to be usable; this is that work. FT4 now fans the band out over slices exactly as FT8 does - `!$omp parallel do schedule(dynamic,1)` over an anchored grid, `dd4` per thread copied from a shared pristine `dd4orig`, per-slice accumulators merged **in slice order** so no float sum depends on which thread finished first, and a period-wide dedupe (`msgseen4`) because slices and the offset pass would otherwise print

the same message twice. Same shapes, same names and the same answers to races as FT8, which was the point: one solution maintained, not two.

```
Measured on the same 240-period hour as item 50, ‘-j 12‘, deadline 1360 ms:

| run | messages | ms/period | share of the deadline |
|---|---|---|---|
| default, single thread | 1716 | 295 | 22 % |
| **default, threaded** | **1715** | **95** | **7 %** |
| 1 member | 1735 | 174 | 13 % |
| 2 members | 1763 | 241 | 18 % |
| 3 members | 1787 | 313 | 23 % (was 75 %) |
| 4 members | 1801 | 383 | 28 % (was 93 %) |
| 5 members | 1813 | 450 | 33 % |
| **6 members** | **1815** | **530** | **39 % (was 131 %)** |

(Measured after the item 52 fixes, which recovered decodes at every threaded setting - the
figures taken before them were 1707, 1774 and 1803.) So six members, which would have been
late every period, now fits in well under half the budget. Two further readings:

- **Five members is the ceiling worth having, not six**: the sixth buys +2 messages for +80 ms.
  Five takes 98 of the 100 the ensemble has to give, for three quarters of six’s cost.
- **Parallelism saturates around 8 threads**, because the anchored grid cuts 100-3300 Hz into
  417 Hz cells and that is **8 slices**, not one per thread. Beyond 8 threads the extra ones
  have no slice to take; 12 threads measured no faster than 8 at any depth.

Two things worth keeping:

- **Slicing loses signals at the boundaries, and the offset pass is how they come back** -
  21 lost against 1 on the ‘ft4hint‘ set, 109 against 16 on the hour (1625 messages against 1715). ‘lft4twopass‘
  therefore defaults **on**, unlike FT8’s ‘FT8TwoSlicings‘; the setting exists so it can be
  switched off for measurement. Both numbers are pinned as test rows.
- **‘threadprivate‘ on a megabyte is a segfault waiting for another machine.** Making
  ‘common/heap8/‘ in ‘subtractft4‘ threadprivate took the static TLS block from 3.54 MB to
  6.72 MB, and glibc takes a thread’s TLS out of the same 8 MB mapping as its stack: at 12
  threads every thread died, FT8’s too, with 0 messages. Raising ‘OMP_STACKSIZE‘ in the GUI
  hid it there and nowhere else. The fix is heap allocatables with threadprivate descriptors
  (TLS back to 3.92 MB) and one shared, build-once copy of the low-pass filter. The suite
  deliberately runs at the default stack so this cannot hide again.

Single-threaded output stays byte-identical to the pre-threading baseline (1716 lines).
```

52. FT4’s threaded output is byte-identical run to run (2026-08-30). Item 51 claimed the threaded message set was reproducible and only the print order varied. Repeatability runs - three runs of each setting on the 240-period hour, comparing whole lines, asked for explicitly - showed that was wrong once the ensemble was on: with 3 members at 12 threads run 2 decoded three messages the others did not and missed two they had; with 6 members the sets matched but two to six lines carried a different SNR or ^ marker.

```
The cause is not a missing lock. A signal on a slice boundary is decoded by both neighbouring
slices, and by the offset pass, at slightly different frequency and DT - 2439 against
2440 Hz in the case that exposed it. The period-wide dedupe kept whichever thread arrived
first, and that copy’s coordinates went into the hint memory; ‘ft4hint_find‘ matches within
3 Hz and 0.1 s, so two periods later a hint was found or missed depending on a race, and the
difference cascaded into whole messages. Plain threading never showed it because the
ensemble is what decodes the marginal signals, and marginal signals are what a boundary
decides.

The fix takes the shared state out of the parallel region instead of locking it harder: each
slice appends its decodes to its own column of ‘ft4res‘, and ‘emit()‘ does the dedupe, the
callback and the hint store afterwards, walking the slices in slice order. Hint consumption
moved there too - it used to retire an entry while other slices were still searching it - with
a threadprivate ‘ft4used‘ mask so a slice cannot spend the same hint twice. With the lists
```

read-only for the duration of the loop the ‘critical(ft4_hints)‘ around ‘ft4hint_find‘ is gone, one less serialisation point.

After it the content is reproducible at every setting measured. The contract is deliberately **the result as a whole, not the print order**: slices finish when they finish, and another machine cuts the band into a different number of slices, so asserting the order would fail spuriously there and would force the emission back inside the loop under a lock - serialising for nothing. The comparison is whole lines, sorted, so it still catches an SNR or a marker that moved. That is stricter than FT8's, which strips SNR and markers before diffing (‘msgkey.awk‘) and would have passed straight through this drift.

The rule worth keeping: **a critical section makes a race safe, it does not make it deterministic.** Anything whose result depends on which thread reaches it first has to be collected per slice and merged in a fixed order, as the float accumulators already were.

53. **The FT4 slice grid is anchored to the frequency scale, not to the band edge** (2026-08-31). Item 51's grid mirrored FT8's cell width - 5000/12 = 417 Hz - and its independence from the thread count, but laid the boundaries out from `nfa` (`nf4lo(k)=nfa+(k-1)*nf4w`) where FT8 places them at absolute multiples of the cell from 0 Hz and takes the cells the decoded range intersects (`slice_count`). The code comment claimed the two were the same; they were not.

The consequence was that FT4's boundaries moved whenever the operator changed the decode range, which is the second half of what FT8's anchoring is for - "a narrower range decodes as the full band does inside it". Measured on 60 periods, deployed build: decoding 100-3300 Hz and 300-3300 Hz gave **30 differing lines in the region both cover**. The first slice also absorbed the remainder, 699 Hz against 417 for the others - the widest span for subtraction to reach across, and the worst-balanced slice in the loop.

‘slice_count‘ now takes the cell width so both modes share it, and FT4 builds its boundaries with FT8's construction. On 100-3300 Hz that is 8 slices at 100-417, 418-834, 835-1251, 1252-1668, 1669-2085, 2086-2502, 2503-2919, 2920-3300, with only the two clipped end slices off the 417 Hz cell; the offset pass shifts every edge by 208 Hz. The same two ranges now give **0 differing lines**. No cost: 94 ms/period against 95, and the hour is unchanged at 1715 messages with 16 lost against a single slice.

What it does not fix, in either mode: **one thread still decodes the whole band in one piece** (‘nsl4=1‘, and FT8's ‘nslicesft8=1‘ with its second and alternate passes gated on ‘numthreads.gt.1‘), so a single-threaded run differs from every multi-threaded one - 6 lost and 3 gained on 60 periods. From 2 threads up the set is identical at every count. Slicing is not a neutral division of work: each slice subtracts into its own copy of the band, so one slice and eight are different decodes, not the same decode reordered.

54. **The seam blind spot in the candidate search** (2026-08-31, from the operator's question about overlapping slices). `getcandidates4` normalises `savsm` by the noise baseline only inside `nfa..nfb`, so its peak loop cannot test the range's own end bins - confirming a local maximum there needs a comparison against an un-normalised neighbour - and ran `nfa+1..nfb-1`. The next slice starts at `nfb+1` and *its* loop at `nfb+2`, so **bins** `nfb` **and** `nfb+1` **were searched by neither slice**: at df = 4.6875 Hz a 9.4 Hz gap at every seam, 66 Hz of a 3200 Hz band. One bin beyond each end is now normalised as well and the loop runs `nfa..nfb`, so every bin is searched exactly once across the slices.

Measured, 12 threads, 100-3300 Hz: with one slicing it recovers **+4 messages** on 60 periods (332 -> 336, three of them the same station in consecutive periods). With the offset pass on - the shipping default - the message set is **identical**, 1715 on the hour before and after, at 92 against 94 ms/period, within noise. The single-thread set is unchanged too.

So the seam effect is real but smaller than the band arithmetic predicted (~4, not ~7), and ‘lft4twopass‘ was already covering all of it. Kept because it is neutral where it does not help and removes the defect rather than relying on a second pass to hide it. The corollary matters more than the fix: **a seam-targeted scheme cannot pay for itself while the offset pass runs**, which retires the narrow-window and overlapping-slice proposals for now - the

offset pass's remaining ~19 gains come from re-decoding the merged band, not from seams.

Then restructured rather than left as a patch. FT4 was already three quarters of the way to FT8's shape: the spectra, the smoothing and the baseline fit were all slice-independent, and the slice entered only in the normalisation and the peak search. ‘getcandidates4‘ now normalises across the baseline's own span - as ‘sync8‘ scores its wide band and lets each slice select from it - and the peak loop scans the slice's range with neighbours that are always valid. Measured results-neutral: identical sets at one slicing, with the offset pass, and at one thread; 1715 messages on the hour at 93 ms/period against 92. The gain is structural - everything before the peak loop is now slice-independent, which is what the sharing optimisation needs.

Next, and the bigger prize: share that band-wide work between slices instead of recomputing it. FT4 repeats ~26 ms of identical spectra per period in the first subtraction pass (~13 % of the plain decode). **FT8 is worse by about 40x, now measured**: ‘sync8‘ is called once per slice (‘ft8_decode.f90:243‘) and computes its sync surface over the *wide* band, 0-5000 Hz, on every call whatever slice it was asked for. 708 ms of CPU per period for the RX phase alone (24 on-air periods, 12 threads, light preset, background off), ~88 ms of wall against a 301 ms RX phase - 29 % of it. Measured per pass rather than assumed: pass 1 is 181 ms, 25.6 %, against 18-19 % each for passes 2-5, so sharing pass 1 is worth ~159 ms of CPU, ~20 ms of wall, ~7 % of the RX phase (~667 ms CPU across the whole period once the background's units are counted, but that runs in idle time). the project notes item 2.1. Neither can change a decode, so both are provable by output equality. the project notes.

Open, measured and written up, not built: the remaining ways to make the slicing cheaper, in the project notes. ‘getcandidates4‘ recomputes the whole period's spectra once per slice, and in the first subtraction pass all eight slices compute the identical set - ~26 ms of CPU per period, ~13 % of the plain decode, for nothing. And the operator's idea of decoding narrow windows on the seams instead of running a whole offset pass: correct in principle, ~2.6x more efficient per millisecond at +-42 Hz, but it recovers about half the messages, because only ~8 of the offset pass's 27 gains are seam-specific and the rest come from re-decoding with a different subtraction history.

Open, and measured the same day: FT8 has the weaker half of this. Asked the same question with ‘test/decode/ft8repeat.sh‘, FT8 keeps its message set across runs but not the SNRs and markers on it - the same message came back at -18 and at -24 dB in two runs - and the pipeline ensemble dropped or gained one marginal background decode in one run of five. It does **not** have any of the four defects above (all verified: ‘cwfilter.f90:72‘ transforms the AP patterns once before any parallel region, and ‘ft8b.f90‘ passes ‘nthr‘ to ‘osd174_91‘ and ‘unpack77‘); what it has is the emission still inside the parallel region, ‘ft8_decode.f90:321-358‘, which is what FT4 moved out. Written up with the evidence, the proposed fix and the reasons it is more delicate than FT4's - it is the production decoder and every expected set moves - in the project notes. Not urgent: the only instability measured costs nothing real.

55. **FT4 split into driver and per-candidate machinery** (2026-09-01, the project notes). The last layout asymmetry: FT8 has always split `ft8_decode.f90` (the slice driver) from `ft8b.f90` (one candidate's decode), while FT4 kept everything in one file - which is why every transplant between the modes needed translation instead of copying. `ft4_decode.f90` now keeps the slice loop, the subtraction passes, `getcandidates4` and `emit`; the new `lib/ft4b.f90` holds the candidate body, the per-thread first-run tables and the a-priori patterns, verbatim. Byte-identical on the 73-period FT4 hour at 1 and at 12 threads (539/537 lines unchanged), `ft4hint.sh` green. One pre-existing oddity became visible at the cut: the /R `chkflscall` rejection RETURNs out of the whole slice decode - skipping the remaining candidates, the remaining passes and the delta collect - rather than just the candidate; preserved exactly (`labort`), noted as a cleanup candidate since it looks like an accident of the single-file layout.

56. **FT4 gets FT8's best-SNR emission rule** (2026-09-01, the project notes). FT4's `emit` kept the first slice's copy of a same-text pair; FT8's `ft8emit` (item 2.2's merge) keeps the better SNR, slice order only breaking ties. Backported: the hint store/consume sweep runs

first, per entry in walk order exactly as before (the store order steers `ft4hint_find`'s first-match in later periods, so it must not depend on the winner), then winner selection with in-place best-SNR replacement, then the commits. Provably inert at one slice (the in-pass `decodes` list blocks same-text repeats within a slice): the 73-period hour is byte-identical at 1 thread, and at 12 threads too - this recording holds no unequal same-pass pair - so the rule is pinned by construction instead: `test/decode/ft4merge.sh` mixes two simulated transmitters sending the same text (slice 3 at -16, slice 4 at -10) and asserts one printed line at -10, where the old rule printed -16. The subroutine is now `ft4emit` (bound as `emit`), naming FT8's `ft8emit`; the `ft4 decode` provenance line (JTDX_FT4_TIMING) gained a `thr` field, which the test reads.

57. **FT4 gets the second near-DT sync peak** (2026-09-01, the last decoding-approach item of the project notes's "FT8 has it" list short of the TX background). FT8's item 2 keeps a second DT peak per bin of its sync surface so two stations on one frequency both get a shot. FT4 has no per-bin DT surface - its candidates are frequency-only and the DT search lives in ft4b, per candidate, where only the strongest coarse peak was ever tried - so the mapping is: the coarse scan tracks its runner-up ($\geq$ 32 samples / ~48 ms from the winner, kept by displacement), and when all three DT segments fail, the runner-up gets one fine-scan-and-decode attempt as a fourth segment (ordinary sweep only; the provenance line shows `iseg 4`). Measured, 12 threads: the 73-period hour 539 $\rightarrow$ 546 (9 gained / 2 lost, both losses displaced repeats still decoded elsewhere; the gains include co-channel catches at -9 and -10 dB); the 240-period night hour **1715 $\rightarrow$ 1748, +33, +1.9 %, for +17 ms/period** - the best gain-per-millisecond in the FT4 menu (one ensemble member costs ~80 ms for +20). Deterministic (two runs identical), and `JTDX_FT4_DT2=0` reproduces the pre-feature output byte for byte. Default on. `ft4hint.sh`'s pinned counts re-recorded for it (+6..+8 a row, losses unchanged in kind; the depth-0 row gains too - the peak helps stock-style decoding as well).

58. **FT4 deep OSD as a real control** (2026-09-01). Item 44 measured `JTDX_FT4_OSDDEEP` as a bad trade and left it an env hook - but that was the single-threaded decoder, and the threading (item 51) plus the second DT peak (item 57) changed the base from 1360 ms-class to 116 ms/period. Re-measured on the 240-period night hour, 12 threads, on top of item 57 (1748 @ 116 ms/period): depth 4 gives 1764 (+16 net, +59/-43 churn) at 291 ms/period (21 % of the deadline), depth 5 gives 1771 (+23, +84/-61) at 720 ms (53 %). So it is affordable now - and still **dominated by ensemble members** (+20 for ~75 ms each, 2-3x cheaper per message), with the largest churn of any FT4 knob. Verdict: shipped as the expert control it deserves to be and nothing more - `lft4deeposd` at the end of the params block (**PARAMS BLOCK CHANGED - jtdx and jtdxjt9 deploy as a pair from here on**), Decode $\rightarrow$ FT4 decoding $\rightarrow$ expert $\rightarrow$ "OSD order 2 for every candidate", ini key `FT4DeepOSD`, default off; the env hook can still force depth 5. `params_layout.sh` green (C and Fortran agree at 328 bytes); no preset uses it, mirroring FT8 where only the retired WeakSignals preset ever did.

59. **FT4 gets the TX background** (2026-09-01, the last item of the project notes's "FT8 has it" list). FT8's pipeline shape carried over: after the RX phase's `<DecodeFinished>` the idle time (the 7.5 s TX window) runs one background unit - the ensemble members the RX phase did not reach (`nft4ensfrom..nft4bgensemble`, ft4b's member ladder continued) - over the retained pristine band (`dd4prist`), both slicings, through the same `ft4emit` merge, so only new messages print and everything feeds the hint memory. Markers as FT8's: `|` for a background decode, the cross for a background hint. Aborts between slicings on the GUI's `.lock` removal or the `.bgabort` sentinel (`bg_lock_gone`, shared with FT8); `nbg4run` mirrors `nbgrun` (1 = GUI rules, 2 = file mode unclocked); the dispatch sits beside FT8's early block and relies on multimode_decoder's blanket `save` for the period's grid state,

as FT8's does. GUI: Decode → FT4 decoding → expert → "TX background members" (off/1..6, the TOTAL member count reached; values at or below the RX count are inert), ini `FT4BgEnsemble`, params `nft4bgensemble` (block change - pair deploys). File mode: `-V n` when `-4`.

```
**Measured, 240-period night hour, 12 threads, RX at the recommended 3 members:**
1811 -> 1855, **+44 background-marked decodes (+46/-2 net, +2.4 %) - all of it in idle
time, zero reply-time cost.** Deterministic, two runs identical. RX at ‘-M 6‘ pays
~530 ms inside the 1360 ms deadline for less; the background gets more for free. Abort
verified (.lock removed 4 s in: ‘<abort>1‘, later periods unaffected). Defaults off; no
preset uses it yet - preset placement after air time. v1 scope: the member unit only;
FT8's classic/residual/retry units follow the same pattern when measured worth while.
**Guards delivered** later the same day - item 60: ‘ft4bg.sh‘, ‘ft4strict.sh‘, the
params-reach check and memcheck's FT4 chain.
```

60. The FT4 TX background guarded (2026-09-01, the project notes 2.5 - item 59 had shipped on hand checks only). Every guard is the mirror of its FT8 counterpart, so the two decoders are held to the same contract by the same kind of test:

- `test/decode/ft4bg.sh` = `bgabort.sh` for FT4, six cases. FT4's unit is ~100 ms plus ~75 ms a member, far too short for a timed `.lock` removal to land inside one period's background as it does inside FT8's 20 s one - so the abort cases run 24 night-hour periods and *hold* the condition for one second (`.lock` removed every 50 ms; `.bgabort` left in place): every period whose background starts inside the window must report `<abort>1` (0 units before the first slicing, 1 between the two), every period before the first aborted one must be byte-identical to an uninterrupted run, the last must run clean, none may be lost, and the run must end within 3 s of the uninterrupted run's time. Measured: 1-2 of 24 abort under the `.lock` window (the RX call in flight stalls in its 1 s `.lock` poll until the window closes), 9-10 of 24 under the sentinel (a period with an aborted background is fast, so more of them fit); +0.2 s and -3.1 s against the uninterrupted run. Plus `ctxstale` (sentinel from the start: every background 0 units, RX still decoding - the P10 regression, never exercised on FT4's path before), `inert` (`-M 3 -V 3` prints no background line and decodes exactly as `-M 3`), `params` and `markers` (3 pipes, 2 crosses, no placeholder over the 24 periods).
- `test/decode/ft4strict.sh` = `ft8strict.sh` for FT4 - FT4's first strict rows of any kind: `-M 0 -V 3 -j 12` (the background on every period), plain `-j 12`, plain `-j 1`, 16 night-hour periods byte for byte against `expected/strict_ft4*.strict`, wisdom hygiene included; the background row also asserts one well-formed status line per period and at least one `|` decode. 103/100/101 lines recorded.
- **Params reach** (`ft4bg.sh` case `params`), which found two real gaps: `JTDX_DUMP_PARAMS` was written from inside the FT8 branch of `multimode_decoder`, so an FT4 run produced no dump at all - now a contained `dump_params()` called from both paths, with `lft4deeposd` and `nft4bgensemble` added to it; and file mode never assigned `lft4deeposd` (the block is zeroed, so `-0` silently applied to FT8 only) - `params%lft4deeposd=ldeeposd` beside the FT8 line. A GUI-side fill bug in either field would now fail this case rather than pass `params_layout.sh` silently.
- `memcheck.sh` **FT4 two-period chain** = the FT8 chain for FT4: the same message 15 s apart (both even periods), -13 dB then -19 dB so the second is a hint decode, `-M 0 -V 3` at 2 threads so the item 59 unit runs its three members under valgrind on both periods; asserts the `^` and two `<units> 1` lines besides the zero-context rule. jtdxjt9 only (no params change); every existing guard re-run green on the new binary. Deployed 2026-09-02 as jtdxjt9 829ebb54 (rollback jtdxjt9.pre-item60-20260902).

61. FT4 prints the same markers as FT8 (2026-09-02, the project notes 2.6 item 1 - the last cosmetic gap of the project notes). ft4b's marker block now carries FT8's rules verba-

tim (ft8_decode.f90 ~368): blank for a plain decode, , / . for free text at / off the QSO frequency, ^ for a hint decode, for any other AP decode (types 1-6, which FT4 had printed unmarked), | / **the cross in the background, and the hint and AP markers take precedence over the free-text ones as in FT8 (FT4 had let , win over ^ on a free-text hint, a case that never occurred on the recorded hours). The special DX-pedition message's 1 was already there; the background now overrides it with | as FT8's does. The GUI side needs nothing:** decodedtext.cpp **already reads** as the low-confidence flag for every mode, so FT4 AP decodes are now flagged in the decode window exactly as FT8's are. On the 16 night-hour periods exactly two lines changed - the two mycall-AP decodes gained the - and nothing else moved (ft4strict.sh diff, expectations re-recorded as the intended change; ft4merge.sh/ft4bg.sh green). ft4hint.sh's line counts were unchanged too, but two of its rows went red for a harness reason: its (period, message) key stripped only the ^, so a message decoded by AP at depth 0 (now) and by hint at depth 4 (^) counted as one lost and one gained - the key now strips both markers and the rows are back to 26/0 and 560/2. jtdxjt9 only; deployed 2026-09-02 as jtdxjt9 8aa16004 (rollback jtdxjt9.pre-item61-20260902).

62. **FT4's /R rejection no longer aborts the slice** (2026-09-02, the project notes 2.6 item 2). JTDX's FT4 decoder answered a chkflscall rejection of a ... R ... message with a RETURN from the whole candidate loop - the remaining candidates of the slice, its remaining passes and the delta collect all skipped for one bad message - where ft8b returns from the per-candidate routine and the next candidate is tried. The item 55 split had preserved that verbatim through a labort flag; the flag is gone and the rejection is now a RETURN from ft4b, which since the split means exactly what ft8b's does. A trace line (ft4 reject /R ... under JTDX_FT4_TIMING) makes the case countable.

Reach, honestly stated: ‘chkflscall‘ is a no-op today - its ALLCALL7 lookup was switched off at source on 2026-08-28 (‘LALLCALL7_FILTER=.false.‘, the July 2024 file was rejecting real QSOs between newly licensed stations), so ‘falsedec‘ is never true and this path cannot fire in production or in any test without flipping that switch. The change is the FT8 mirror for the day the lookup returns, and the measurement is accordingly a null one: every FT4 row byte-identical (‘ft4strict.sh‘ three rows, ‘ft4hint.sh‘ all pinned numbers, ‘ft4merge.sh‘, ‘ft4bg.sh‘), no re-pinning needed - the "re-pinning it would need" of the TODO entry was written before the switch was checked. jtdxjt9 only.

63. **One thread ladder** (2026-09-02, the project notes 2.6 item 3). The rule that turns "threads = auto" into a number (1 core → 1; up to 4 cores leave one free, up to 8 two, up to 15 three, up to 20 four, up to 29 five; 30+ capped at 24) and applies an explicit setting existed three times by hand: JTDX's inside decoder.f90's FT8 branch, a copy in the FT4 branch (needed because the FT8 one sits inside its branch and the variable was still zero when FT4 ran - the first FT4 slice loop silently never ran for that reason), and auto_ft8_threads()/effective_ft8_threads() in decodepreset.h, which the GUI needs before the decoder runs to size a preset's member count. Now: src/lib/thread_ladder.f90 (auto_threads, decoder_threads), called from both mode paths; the C++ copy stays as the GUI's and params_layout.sh **pins it against the Fortran over every setting 0..24 and every core count 1..40** (1000 pairs), the way it pins the params block. One real divergence surfaced and was settled the GUI's way: for a setting of 25 or more JTDX's FT8 copy fell back to ONE thread, where the FT4 copy and the C++ clamped to the core count - the GUI never sends such a value (readSettings clamps to 0..24), file mode's -j 25+ did, and it now clamps like the others. Nothing else changes on any machine: every guard byte-identical. This machine: 16 cores, auto = 12 threads. jtdxjt9 only; deployed with item 62 on 2026-09-02 as jtdxjt9 97e9abad (rollback jtdxjt9.pre-item63-20260902).

64. **An FT4 full-band benchmark, synthesised from the FT8 one** (2026-09-02). Every FT4 number so far came from the WW Digi hours - a few signals a period. FT4 had no equiv-

alent of `wav/full_band_16.wav`, the crowded 0-5000 Hz FT8 period behind every FT8 recipe, so `test/decode/ft4_from_ft8.py` makes one from it: the capture's 104 validated messages (`reference_set.tsv`, median frequency/SNR/DT per message over every decoder run), each generated CLEAN by `ft4sim` (SNR 99: no noise, unit amplitude at a 1e8 peak), scaled by ft4sim's own rule sig = sqrt(2*2500/6000) * 10^(snr/20), summed, one seeded unit-variance noise realisation, ft4sim's 2.7e7 gain - so the levels mean exactly what a single ft4sim file's do (summing noisy files would have lifted the floor by 20 dB). SNR and DT unchanged (both sims reference 2500 Hz; the DTs span -0.6..+0.8 s, inside FT4's window); frequency stretched by the signal-width ratio 83.3/50 = 1.667, so every pair keeps its separation IN SIGNAL WIDTHS and the capture's 0-3050 Hz fills 0-5083 Hz; the one signal above the band (5B4ALJ JA0UUA at 3050 → 5083 Hz) is dropped. **103 signals, every one verified to decode alone at -5 dB** - which the first cut of the file could not claim: three of its signals above 4870 Hz could never decode, because JTDX's FT4 candidate search stopped at 4910 Hz on the sync bin. That became item 65; until it landed the file was cut at 4850 Hz with 99 signals (default 74, `-M 6` 76, six passes 75, 6-cell 74, x3 76). Manifest `wav/ft4_full_band.tsv`, 16- and 32-bit files identical to the decoder.

```
**Calibration:** the reported SNR sits ~0.9 dB below the intended on average (sd 2.9, the
two decoders' estimators), DT matches to 0.00 s, frequencies 130..4850 Hz as laid out.
Run to run byte-identical; the 16-bit copy decodes the same lines.

**What it shows (12 threads, 50-5000 Hz, the final 103-signal file):** default recipe
**71 of 103, zero false**; the 32 misses are 14 of the 17 signals below -15 dB (FT4's floor
is 3.5 dB above FT8's - physics) and 18 of the 47 in -15..-11 dB, none above -10. And the
knobs move it: `-M 6` 77 (0.31 -> 0.88 s), six subtraction passes 76, a 6-cell grid 75,
1 thread 67; three identical periods 15 s apart stay at 71 (the 99-signal cut had gained
two there). The FT4 recipes were tuned on
the sparse hours (items 48/49); this file says the pass count and the slice width matter
on a crowded band, which the recorded hours could never show - the project notes 2.7.
`test/decode/ft4bench.sh` pins the seven rows, each with zero false decodes.
```

65. **FT4 decodes to the top of the band, as FT8 does** (2026-09-02, found by item 64's file: three of its signals could never decode). **Measured alone at -5 dB, range 50-5000 Hz:** FT8 decodes a signal at 4900, 4940, 4950 and 4960 Hz; FT4 decoded at 4870 and not at 4875. The cause is JTDX's `getcandidates4` (inherited from WSJT-X), which clamps both the decode range and the sync-bin search at 4910 Hz; an FT4 candidate's sync bin sits 31.25 Hz above the signal's base frequency (`f_offset = -1.5*12000/NSPS`), so the last base frequency that got a candidate was ~4875 Hz - the GUI's 5000 Hz decoded to 4875 silently. FT8's sync8 scores the wide band to nfbwide = 5000 Hz and has no such cut. **The lower edge is equal:** both modes decode a signal at 20 Hz alone with the range set from 0.

```
The fix keeps every existing decode bit-identical: the baseline is still **fitted** over
JTDX's 200-4910 Hz (`nwb`), so the polynomial and every normalised bin below 4910 Hz are
exactly what they were; it is now **evaluated** to 5000 Hz (`ft4_baseline` gained an
evaluation-bin argument `nfe`, its `.org` taken first - the file was pristine), and the
normalisation, the peak search and the range clamp run to 5000 Hz (`nwe`), so a base
frequency up to ~4969 Hz gets its candidate as an FT8 one does. The downsample takes its
window around f0 in the full 6 kHz spectrum, so nothing else bounds it. Below 4910 Hz the
FT4 strict rows, ft4hint's numbers and the merge test must come back byte-identical -
that is the acceptance, together with the ceiling probe 4850-4990 Hz and item 64's file
regenerated with the cut at 4960 Hz (three more of the FT8 capture's messages fit).
**Measured on the built decoder (jtdxjt9 11c13654):** alone at -5 dB the ceiling probe
decodes at 4850, 4900, 4940, 4950, 4960 and **4969 Hz** and not at 4975 - the 5000 Hz
sync bin less 31.25 Hz, exactly as designed, against 4870/4875 before; the FT4 strict
rows, ft8strict, ft4merge and ft4bg byte-identical / green; item 64's file regenerated
with the cut at 4960 Hz keeps **103 of the 104** messages, all 103 verified alone,
`ft4bench.sh` re-pinned (default 71, 1t 67, `-M 6` 77, six passes 76, 6-cell 75, x3 71).
jtdxjt9 only; deployed 2026-09-02 as jtdxjt9 11c13654 (rollback jtdxjt9.pre-item65-20260902).
```

66. **The FT4 recipe re-tuned on the crowded file: four subtraction passes** (2026-09-02, the project notes 2.7). The sweep item 64 asked for - passes 2-6 x grid cells 6/8/12 x members 0/3/6, then deep OSD, the alternate pass, the candidate cap and OSD depth 5 on the leaders, 61 runs on `wav/ft4_full_band_16.wav` at 12 threads, 50-5000 Hz, every row zero false - and then the sixteen leading recipes on both recorded hours (73 + 242 periods) against the current default, counting (period, message) pairs gained and lost and the RX wall time.

```
**The crowded file (true of 103 / RX s):** default 3 passes 12 cells 0 members **71 /
0.27**; 4 passes **76 / 0.32**; 5 and 6 passes 76; 3 members 76 / 0.54; 6 members 77 /
0.82; 8 cells with 6 members 79 / 1.45 (4-6 passes alike); 6 cells 75 / 0.30 at 0
members; 24 cells 64. Deep OSD: +2 at 0 members, +0 at 6, for 2x the time. The alternate
pass: +0 either way. The most anyone got was 81, with 4 cells, 6 members, deep OSD and
alt - at 11 s.

**The recorded hours (messages, gained/lost vs the default, mean/max RX s):** 4 passes
is a **null change everywhere** - 546 and 1748 messages, +0/-0 on both hours at 0, 3 and
6 members, the three strict rows byte-identical, +0.01-0.04 s. Members: 3 -> +12/-2 day,
+74/-11 night at 0.33 s; 6 -> +15/-1, +108/-14 at 0.55 s (max 1.06). 8 cells: +1/-0 day,
+15/-5 night at 0 members, +2..+3 net with members, +0.1-0.2 s. **The operator's live
configuration** (6 members + deep OSD + alternate pass, the production ini): 563 and
1858 messages, +19/-2 and +123/-13 - the alternate pass and deep OSD DO add ~16 on the
sparse night hour - **at a mean RX time of 2.05-2.31 s and a maximum of 4.35-4.97 s
against the 1360 ms reply deadline** (file mode's full 0-5000 Hz; the GUI's range is
narrower, the overrun smaller but real). 3/12/6 alone gives 1842 at 0.55 s mean.

**Decisions:** (1) `nft4nsp` 3 -> 4, the default - +5 on the crowded band for 50 ms and
provably nothing on the recorded hours (`ft4strict` three rows and `ft4hint`'s 545/546
unchanged, pinned). (2) The grid stays at 12 cells: the 8-cell gains are modest, come
with losses, and the 417 Hz cell is the one FT8 shares - a candidate for a later
crowded-band mode, not a default. (3) The presets keep their member counts, which the
sweep confirms dominate everything else per second spent. (4) A note for the operator,
not a code change: RX 6 members + deep OSD + alt overruns the deadline on this machine;
RX 3 (the "recommended" preset) with the TX background at 6 gets the members' gain in
idle time at 0.33 s. `ft4bench.sh` re-pinned on the new default (76, 1t 70, `-M 6` 77,
six passes 76, 6-cell 75, x3 76). jtdxjt9 only; deployed 2026-09-02 as jtdxjt9 e5170cff
(rollback jtdxjt9.pre-item66-20260902).
```

67. **FT4 presets in FT8's tiers** (2026-09-02). The four FT4 presets of item 51 knew three knobs (effort, alternate pass, RX members) and were calibrated before deep OSD (item 58) and the TX background (item 59) existed. Both recorded hours were re-measured with the two new knobs on the four-pass default (idle machine, 12 threads, file mode's 0-5000 Hz):

```
| recipe | night (242 periods) | +/- vs default | day (73) | reply time mean / max |
|---|---|---|---|---|
| default: deep, 4 passes | 1748 | - | 546 | 0.12 / 0.22 s |
| background 3 | 1826 | +82 / -4 | 557 | 0.12 / 0.22 |
| background 6 | 1856 | +114 / -6 | 559 | 0.12 / 0.22 |
| deep OSD + background 6 | **1862** | +126 / -12 | **565** | 0.27 / 0.65 |
| 3 members + background 6 | 1855 | +118 / -11 | 560 | 0.34 / 0.75 |
| 3 members alone | 1811 | +74 / -11 | 556 | 0.34 / 0.75 |
| 6 members alone | 1842 | +108 / -14 | 560 | 0.57 / 1.29 |
| 3 members + OSD + bg 6 | 1859 | +123 / -12 | 565 | 0.82 / 2.06 |
| 3 members + alt + bg 6 | 1861 | +123 / -10 | 562 | 0.61 / 1.64 |
| 6 members + OSD | 1849 | +116 / -15 | 561 | 1.37 / 3.56 |
| 6 members + OSD + alt (the production ini) | 1858 | +123 / -13 | 563 | 2.47 / 6.02 |

**The background does the members' work for free:** six members in the TX window beat six
at reply time (1856 against 1842) at 0.12 s instead of 0.57 - and deep OSD on top, cheap
at zero members, is the best recipe on both hours at 0.27 s. Every reply-time member is
dominated. So the new set keeps the RX phase light and spends in the background, in FT8's
```

```
tiers with FT8's dots and colours (‘ft4_preset_colour‘), the recommended one underlined:

| tier | recipe (effort / alt / RX members / deep OSD / background) | gain | reply time |
|---|---|---|---|
| fast | 1 / - / 0 / - / 0 | | |
| default | 3 / - / 0 / - / 0 | | 0.12 s |
| best power (grey) | 3 / - / 0 / - / 3 | +4.5 % | 0.12 s |
| best value (orange) | 3 / - / 0 / - / 6 | +6.2 % | 0.12 s |
| **recommended: best results** (green) | 3 / - / 0 / OSD / 6 | +6.5 % night, +3.5 % day | 0.27 s |
| most at reply time (blue) | 3 / - / 3 / - / 6 | +6.1 % | 0.34 s |
| max effort (red) | 3 / alt / 6 / OSD / 0 | +6.3 % | 2.5 s - over the deadline |

"most at reply time" is the one tier that still spends members in the period, because the
message a QSO is waiting for is decoded there and the background only ever adds late; "max
effort" is the RX-only maximum, kept as FT8 keeps its "ensemble" preset - no time budget -
and labelled with its overrun (item 66 found it in the production ini). The old 3- and
5-member RX recipes are Custom now. Mechanics: ‘FT4Recipe‘ gained ‘deeposd‘ and ‘bg‘, the
Custom detection compares all five with the inert-background rule (a target at or below
the RX count compares as off, since it runs nothing - item 59), ‘applyFT4Preset‘ sets the
two new controls, ‘markFT4Presets‘ marks the submenu as ‘markRecommendedPresets‘ does, and
the presets submenu carries the green dot. ‘test/harness/test_preset.cpp‘ covers every
recipe, the map-back, the inert rule and the retired recipes.

**On the crowded file** (item 64, 103 signals, 50-5000 Hz; true / of which background /
reply time): fast 33 / 0 / 0.13 s, default 76 / 0 / 0.32, best power 76 / 0 / 0.34, best
value 77 / 1 / 0.32, recommended 77 / 4 / 0.56, most at reply time 77 / 1 / 0.76, max
effort 77 / 0 / 4.01 - zero false everywhere. Every tier above default sits at the
members' ceiling of 77 for that file: with four passes the RX phase already takes what
members can add there (+1, against +6 on three passes in item 64), and what is left sits
in the -15..-11 dB band that only a wider grid reaches (8 cells 79, item 66). So on a
crowded band the tiers differ in cost, not yield; on the sparse hours they differ in both.

**Seven noise realisations** (‘ft4_from_ft8.py --seed 7..13‘, the same 103 signals under
different noise - one realisation moves any recipe by +-3, and the seed-7 file alone had
deep OSD at -3 and 8 cells at +3 where seed 8 had them at +3 and -1). Means over the
seven, true of 103 / at reply time (total less the background's late decodes) / reply
time; default 68.3 / 68.3 / 0.29 s: background 3 **70.7 / 68.3 / 0.30**; background 6
**72.0 / 68.3 / 0.30**; deep OSD + background 6 71.9 / 68.3 / 0.58; 3 members + bg 6
71.7 / 69.9 / 0.64; 6 members 71.1 / 71.1 / 1.02; 6 members + OSD + alt 71.3 / 71.3 /
3.78; deep OSD alone 68.3 / 68.3 / 0.58; 8 cells + bg 6 72.6 / 66.9 / 0.34; 8 cells
alone 66.9. So on a crowded band: **deep OSD adds exactly nothing** (68.3 = 68.3, +0.28
s), the background does all the gaining at no reply-time cost, six reply-time members
are the most at reply time (+2.8, 1.02 s) and max effort buys nothing beyond them for 4x
the time; the 8-cell grid has the best total and the worst reply time. Members produced
one false decode in seven runs in three of the member recipes, the background none.

**The cell width on the production ranges** (same seven seeds, 100-3100 and 50-3600 Hz,
divisors 4-24 = 1250-208 Hz cells, four recipes): a plateau. From 5 to 16 cells every
recipe's total sits within +-1 of its 12-cell figure on both ranges (38-39 default / 42-43
bg6 / 41-43 m6 on 100-3100; 45-46 / 48-50 / 48-50 on 50-3600), which is the noise of seven
realisations; only 24 cells (208 Hz, 2.5 signal widths) loses ~2 everywhere, and only 4-5
cells cost reply time (0.53 s against 0.25, fewer and longer slices). Reply time falls
steadily with more cells; yield does not move. So the 417 Hz cell FT8 shares is as good
as any for FT4 on a crowded band, the seed-7 "+3 at 8 cells" of the first sweep was one
realisation, and a crowded-band grid mode is not worth building. jtdx only - no params change;
deployed 2026-09-02 as jtdx aedc2dcf (rollback jtdx.pre-item67-20260902).
```

68. **The FT4 presets after the crowded-band check: six, not seven** (2026-09-02). Item 67's seven-seed ranking on the crowded file changed two recipes and merged two tiers:

- **recommended is background 6 alone** - deep OSD, which the first recommended recipe carried, adds nothing on a crowded band at reply time or in total and doubles the reply time (0.58 against 0.30 s), and on the sparse hours it was worth 6 of 1856. Background 6 is the

recipe that is never worse at reply time on either file and within noise of the best total on both, so it carries the "best value" mark and the recommended underline; the separate "best value" entry is gone.

- **most at reply time is six members in the period, no background** - +2.8 at reply time on the crowded file against +1.6 for 3 members + background, and 1842 against 1811 on the night hour; 0.57 s mean, 1.29 s worst period, inside the deadline; the label carries the one false decode in seven crowded runs that member recipes produced. Best power (background 3), fast, default and max effort stay. Deep OSD is now set by no preset but max effort, so a stored `FT4DeepOSD=true` with background 6 shows as Custom. `test_preset` covers the new recipes, the map-back and the inert rule. jtdx only.

69. **The FT4 decoding menu in FT8's shape: an RX submenu and a TX background submenu, each phase with its own settings** (2026-09-02). FT8's expert menu has had the two phases side by side since the pipeline (RX: cycles, sensitivities, members, OSD, slicing, alt; TX background: the same again for the idle-time decode). FT4's background (item 59) could only set its member target and otherwise inherited whatever the RX phase had switched on. Now Decode → FT4 decoding → expert → **TX background** carries the phase's own effort (fast/medium/deep), ensemble effort (the member target, moved here from RX), the alternate pass, deep OSD and the second slicing pass, and the RX submenu keeps the reply-time set. **PARAMS BLOCK CHANGED** - four fields appended, `nft4bgdepth` (0 = the RX phase's), `lft4bgdeeposd`, `lft4bgaltpass`, `lft4bgtwopass`; 332 → 340 bytes, `params_layout.sh` green, jtdx and jtdxjt9 deploy as a pair. Ini keys `FT4BgDepth` (3), `FT4BgDeepOSD`, `FT4BgAltPass` (false), `FT4BgTwoSlicings` (true); file-mode hooks `JTDX_FT4_BGDEPTH`, `JTDX_FT4_BGOSD`, `JTDX_FT4_BGALT`, `JTDX_FT4_BGTWOPASS`, pinned through `JTDX_DUMP_PARAMS` in `ft4bg.sh`'s params case. In the decoder `ft4_background` sets `nft4alt`, `nft4osddeep`, the depth and the slicing from the bg fields, starting from the env hooks' base values (`nft4osdbase`/`nft4altbase`, ft4_mod1) rather than from the RX phase's switches. The presets' recipes gained the four bg-phase fields (all at the RX defaults: deep, no OSD, no alt, second slicing on - until the per-phase extras are measured) and the Custom detection compares them only while the background is active. The three FT4 submenus now carry FT8's colours (presets green, RX blue, TX background red).

```
**A latent defect found on the way:** ‘nft4osddeep‘ was only ever *raised* by the deep-OSD
switch (item 58) and never reset, so deep OSD switched off in the GUI stayed at order 2
until a restart; it is now reset to its base at every decode. Defaults chosen so nothing
moves: the three FT4 strict rows byte-identical, ‘ft4bg.sh‘, ‘ft4bench.sh‘, ‘ft4merge.sh‘,
‘ft8strict.sh‘ and the preset harness green; ‘ft4hint.sh‘ and ‘memcheck.sh‘ green on the pair.

**What the phase's own settings are worth** (both recorded hours, RX phase identical in
every row - 0.12 s reply time - against background 6 alone at 1856 night / 559 day):
background + alternate pass **1864 (+9/-1) / 560**, 1.7x the background's CPU; + deep
OSD 1866 (+32/-22) / 564, 2.3x; **+ deep OSD + alternate pass 1875 (+41/-22) / 565**,
3.8x - the best night-hour figure of the day, +19 net for nothing at reply time; medium
effort in the background 1757 (+7/-106) - the members' decodes come through OSD, so a
shallower background loses them; without the second slicing 1837 (+5/-24); background 3
with both extras 1854 (+33/-35) / 563 - two extras on three members equal six plain
members at 2x the CPU. So the extras belong in the background and nowhere at reply time
(item 68 measured them there), and the alternate pass is the clean one (one loss). The
preset recipes are unchanged by this item; the follow-up decides whether "recommended"
takes the two extras in its background phase. Deployed 2026-09-02 as the pair jtdx 63ce3f25
/ jtdxjt9 62920ec5 (rollback pair *.pre-item69-20260902, matching halves only).
```

70. **"recommended" takes deep OSD and the alternate pass in its background phase** (2026-09-02, the follow-up item 69 left open). Recipe: effort deep, no alternate pass, no RX members, no deep OSD at reply time, background 6 with deep OSD and the alternate pass, second slicing on - 1875 on the night hour (+7.3 % over the default, +19 over the plain

background) and 565 on the day hour (+3.5 %), reply time 0.12 s, about 0.9 s of idle CPU per 7.5 s period on 12 threads. "best power" stays a plain background 3: three members with both extras only equal six plain members at twice the CPU. Label, tier mark and the harness updated; a stored plain background 6 shows as Custom now. jtdx only, params unchanged (340 bytes) - pairs with the deployed jtdxjt9 62920ec5. Deployed 2026-09-02 as jtdx de2a0950 (rollback jtdx.pre-item70-20260902).

71. **Two FT8 features measured for FT4 before building: the residual unit and a sensitivity knob** (2026-09-02). FT8's second ten percent came from background units FT4 lacks and from its "decoder sensitivity"; both were tried on the crowded file (seven noise realisations, item 64) and on both recorded hours.

- **The residual unit**, prototyped behind `JTDX_FT4_RESIDUAL=f` in `ft4_background` (FT8's shape: after the members, every known decode subtracted - the merged deltas - the sync threshold scaled by f, no members, one more slicing over the band; byte-identical with the hook unset, `ft4strict` confirmed). On top of the recommended recipe: night hour **1888 against 1875, +13/-0** at f = 0.8, 0.6 and 0.5 alike, ~0.12 s of idle time a period; day hour 565 = 565; crowded file 72.2 = 72.2 at every factor. A clean +0.7 % on the sparse night, nothing anywhere else - the crowded misses are below FT4's floor, not behind a threshold. Kept as the hook; making it a TX-background switch is one bool in the params block (a pair deploy) whenever the next block change comes.
- **Sensitivity**: the candidate sync minimum (`JTDX_FT4_SYNCMIN`, 1.2) and the sync-quality gate (`JTDX_FT4_SYNCQUAL`, 20), 16 combinations. Crowded file: lowering either LOSES - syncmin 1.0 or 0.8 67.4 against 68.2 at +20 % reply time, syncqual 24 64.5; syncqual 12 68.7 (+0.5, noise). Recorded night hour: syncmin 1.0 **+15/-1**, 0.8 +19/-1, syncqual 16 +5/-0, syncmin 1.0 + syncqual 16 **1769 against 1748, +22/-1 (+1.2 %)**; day hour unchanged in every row. So FT4 does have a small sparse-band sensitivity knob - the exact shape of FT8's "decoder sensitivity 2", which was +3 % there - worth +1.2 % at +20 % reply time and worth -1 on a crowded band. Left as hooks: not a params change on its own; bundle a `nft4sens` field with the residual switch if either is built.

72. **Both built, one params change** (2026-09-02). **PARAMS BLOCK CHANGED, 340 → 344 bytes**: `lft4bgresidual` (the residual unit in the FT4 TX background) and `nft4sens` (FT4 decoder sensitivity: 0 JTDX's thresholds, 1 sync minimum 1.0 + sync quality 16) appended; `params_layout.sh` green; jtdx and jtdxjt9 deploy as a pair. GUI: Decode → FT4 decoding → expert → RX → **decoder sensitivity** (standard / low thresholds, FT8's submenu shape) and → TX background → **residual unit**; ini keys `FT4Sensitivity` (0) and `FT4BgResidual` (false); file-mode hooks `JTDX_FT4_SENS=1` and `JTDX_FT4_BGRESIDUAL=1` (`JTDX_FT4_RESIDUAL=f` stays as the factor experiment), pinned in `ft4bg.sh`'s params case. Decoder: the sync minimum and the sync-quality gate are now reset at every decode like the OSD depth (item 69's fix, same latent stickiness), the switch lowers them to 1.0 / 16 and the env hooks can still go lower; `ft4_background` runs the residual slicing when the switch is on with FT8's 0.8. Presets: `FT4Recipe` gained `sens` and `bgresidual`; **recommended turns the residual unit on** (night hour 1888, +8.0 % over the default, day 565, reply time 0.12 s, ~1 s of idle CPU a period), no preset sets the sensitivity - the operator's call for a sparse band. Defaults chosen so nothing moves: the three FT4 strict rows byte-identical, `ft4bg`, `ft4bench`, `ft4merge`, `ft8strict`, the preset harness green; `ft4hint` and `memcheck` on the pair below. Two properties worth knowing: the background phase inherits the RX phase's sensitivity (it starts from the RX thresholds, the residual scales from there), and the residual unit runs only while the background runs - a target at or below the RX member count is inert, so the switch is inert with it; the preset matcher treats it that way and both menu labels say so.

73. **Four more FT8 things for FT4, one params change** (2026-09-02). **PARAMS BLOCK 344 → 348 bytes** (`nft4bgsens`), pair deploy.

- **The TX background's own sensitivity.** Item 72's split was incomplete: the background inherited the RX thresholds. Now the background phase starts from the hooks' base values (`ft4syncminbase`/`nft4syncqbase`, captured after every hook) and applies its own switch, and the residual unit scales from that; the RX submenu's label no longer says "the background inherits it". GUI: TX background → decoder sensitivity; ini `FT4BgSensitivity` (0); hook `JTDX_FT4_BGSENS=1`. No preset sets it (measured below).
- **Members by thread count.** FT8's `ensemble_members()` ladder, for FT4: an **auto** entry in both ensemble menus - RX 6 members from 12 threads, 4 from 8, 3 from 6, 2 from 4, 1 from 3, none below; background at least 3 from 3 threads, the clock cutting what does not fit. `ft4_members_auto`/`ft4_bg_auto` in `decodepreset.h` and in `thread_ladder.f90` (file mode's `-M auto` / `-V auto`; the typed word is -3 internally, since -1 is FT8's default that already means auto and FT4 must not run a background unless asked - the first build did, and the strict rows caught it), pinned against each other by `params_layout.sh` over 1000 (setting, cores) pairs. The presets use auto where the count should follow the machine: recommended's background and most-at-reply-time's members (both 6 here); best power keeps a fixed 3, max effort a fixed 6. The preset matcher resolves auto on both sides at the current thread count, so on a 6-thread machine the same preset is recognised with 3 members. `FT4EnsembleEffort=-1` in the ini is auto now; readSettings' "no key yet" sentinel moved to -99.
- **A clock on the background.** FT8's rule in `ft4_background`: under the GUI's rules (`nbg4run=1`, `JTDX_BG_LOCK=1` in file mode) a slicing is not started when the previous slicing's time would not fit before the deadline - the 7.5 s period, or the `-k` budget, from the decode's start less the margin. File mode stays unclocked. The status line gained FT8's `<secs>` field (the GUI parser already read it; `ft4strict`, `ft4bg` and `memcheck` regexes updated, and `ft4bg`'s prefix compare strips it as it strips `<rxs>`), units count only when every slicing ran. `ft4bg.sh` gained a `clock` case: `-k 3` ends every background at once (0 units, no abort), `-k 300` runs them all.
- **The preset lamp reads the FT4 preset in FT4 mode**: letters F fast, 4 default (3 in FT8; renamed 2026-09-05), P best power, R recommended, O most at reply time, M max effort, "Custom"; FT8's colour scheme through `ft4_preset_colour`; the FT4 RX and TX submenus are hooked to the lamp as FT8's are and `refreshFT4Preset` refreshes it; greyed only in other modes. Defaults chosen so nothing moves: the three FT4 strict rows byte-identical; `params_layout` (348), the preset harness (ladders, auto resolution, the lamp letters), `ft4bg` (all seven cases), `ft4bench`, `ft4merge`, `ft8strict` green. Built as the pair jtdx 75e238fe / jtdxjt9 cad5e101; `ft4hint` and `memcheck` green on it.

```
**The background's own sensitivity, measured** (both hours, on top of the recommended
recipe): night 1895 against 1888 (+8/-1), day 565 = 565, the background's wall time 2.54 ->
3.22 s a period on 12 threads; with the RX switch on as well 1894 (+8/-2). +0.4 % for 27 %
more idle CPU - left out of every preset, the operator's call. Worth knowing from the same
run: the recommended background takes **2.5 s of the 7.5 s period on 12 threads** (four
slicings, OSD and the alternate pass in each) - on a 6-thread machine that is ~5 s and on 4
threads more than the window, which is exactly what the clock is for. Deployed 2026-09-02
with item 72 as the pair jtdx 75e238fe / jtdxjt9 cad5e101 (rollback pair
*.pre-item73-20260902, matching halves only - the 340-byte item 69/70 pair).
```

74. **The FT4 TX background's explicit switch, and the presets set every control** (2026-09-02). FT8's TX background submenu opens with "TX background decoding" - an explicit on/off that the presets set (`DecodeRecipe::background`); FT4's background was on whenever its member target exceeded the RX count. Now the same switch heads FT4's submenu: `m_ft4BgEnabled`, ini `FT4BgEnabled` (no key yet = on when a member target was stored, item 59's rule), off passes a zero target to the decoder, so no params change. The

recipe gained `bgon` and also `twopass` - the RX second slicing pass, which FT8's recipes had always set per phase and FT4's presets had left to the operator - so a preset now sets all fourteen FT4 controls, as `applyDecodePreset` sets FT8's; the matcher treats the switch off as a background off (a stored target is inert with it) and the RX second slicing off as Custom. Audit, every FT4 setting: read from the ini, written to it, passed in the params block and set by `applyFT4Preset` - all fourteen, one line each. jtdx only; deployed 2026-09-02 as jtdx 63d56253 (rollback jtdx.pre-item74-20260902).

75. **The virtual candidate at the QSO frequency - FT8's "QSO RX freq sensitivity" and the QSO-side memory behind it, for FT4** (2026-09-02). The one FT8 mechanism aimed at reply time rather than at the band: when the reply a QSO is waiting for has no candidate of its own (masked by a stronger neighbour, or under the sync threshold), FT8 decodes at the QSO frequency anyway, at the DT the partner was last heard at, with the QSO's a-priori bit patterns (sync8 ~318 appends two virtual candidates; ft8b ~85-135 takes their DT from the last message received or the per-call DT list). FT4 had neither the memory nor the candidate. Now, all in FT8's shape:

- **The memory** (`ft4_mod1`): per parity the DT at which each station was last heard (150 deep, the sender = the second call, filled at emit in emission order), and the last message addressed to me by the DX call with its DT; cleared with the hint lists. Once per period `ft4qso_seed` (FT8's rules: DX call set, last TX one of Tx1-Tx4, the hint pass not stopped) derives the DT to try: the last RX message's, else at level 2 the per-call list's, else the hint lists' last message carrying the DX call.
- **The candidate**: on the first subtraction pass, in the slice holding the QSO frequency, one candidate is appended there; `ft4b` syncs it in one window around that DT (+-0.12 s, +-0.25 at level 3) instead of the three-segment search, **trusts the DT instead of a sync score** (the coarse minimum of 1.2 and the sync-quality gate of 20 do not apply - the first build applied them and the candidate never survived a strong neighbour), and decodes it with OSD depth 4 (5 at level 3) - the deeper OSD near the QSO frequency that JTDX's own FT4 code left commented out.
- **Levels** (0 off, 1 low: the last message's DT only, 2 medium: also the per-call list, 3 high: also the wider window and OSD 5), per phase: `nft4rxfsens` / `nft4bgrxfsens`, **PARAMS BLOCK 348 → 356 bytes**, pair deploy; RX and TX background submenus "QSO RX freq sensitivity" low/medium/high as FT8's; ini `FT4QSORXfreqSensitivity` / `FT4BgQSORXfreqSensitivity` (medium); file mode `-R` / `-Z` as for FT8, plus the QSO state file mode never had: `JTDX_LASTTX=0..6` and `JTDX_QSOPROGRESS=0..5`. Every preset carries medium in both phases, as FT8's do. A `ft4 virt pass` trace under `JTDX_FT4_TIMING` shows each pass on the candidate.

```
**Measured** on a simulated QSO (the partner's report at -8 dB in period 1, the reply
"CE3TSK W9XYZ R-15" at the same frequency and DT in period 2, seven noise realisations,
decodes of 7 at level 0 / 2 / 3): alone -17: 6/6/6, -18: 3/4/4, -19: 2/3/3, -20: 1/1/1,
-21: 0; under a -5 dB neighbour 40 Hz above: -17: 5/6/6, -18: 4/4/4, -19: 1/2/2, -20: 0.
So about **+1 dB on the reply's threshold and the recovery of a candidate the neighbour
masks**, zero false decodes in 294 runs, level 3 no better than 2; the trace shows the
mycall+DX-call a-priori pass carrying every gained decode and the LDPC/OSD giving up
below -20 dB, which is where FT4's 4.5 s signal runs out of energy (FT8's virtual path
reaches ~-24 with 12.6 s). Smaller than FT8's gain, for that reason, but it is the
reply-time decode a QSO waits for. ‘test/decode/ft4qso.sh‘ pins three realisations where
level 0 fails and level 2 decodes, zero false, and that the mechanism is byte-identical
to off when no QSO is in progress (file mode's default); the FT4 strict rows, ‘ft4bg‘
(params case now covers the two levels), ‘ft4bench‘, ‘ft4merge‘, ‘ft8strict‘, the layout
and the preset harness green. Built as the pair jtdx 37c9485b / jtdxjt9 eead0d91.
With it the last hint-data difference of the project notes is closed: FT4 keeps what
FT8 keeps; only the hint *test* differs (FT8's matched filter against FT4's a-priori
pass), by design. **Follow-up, both modes (the project notes 2.8):** the deeper OSD carried every
```

gained decode here, and JTDX's FT8 pulls the OSD back to depth 3 on the QSO-frequency candidates while a QSO is in progress (‘ft8b.f90‘ ~1518, "unload CPU, let ft8s pick up QSO message") - it relies on the hint decoder there, which needs the partner's previous message and never matches a new report. Extra effort on that one candidate costs almost nothing at reply time and the background can spend far more on it when the RX phase failed; the analysis to size it, for FT8 and FT4 alike, is written up as 2.8. Deployed 2026-09-02 as the pair jtdx 37c9485b / jtdxjt9 eead0d91 (rollback pair *.pre-item75-20260902, matching halves only). Tests completed after the deploy, same day: (item 76 follows below this entry's test note)
‘ft4qso.sh‘ gained the per-call-list source (level 1 seeds nothing, level 2 seeds from source 2 and decodes) and the background phase's own level (RX 0, background 2: the reply prints as a background decode); ‘memcheck.sh‘ gained an FT4 QSO case so the new paths run under valgrind.

76. **sync8's pass-1 surface shared across the slices** (2026-09-02, the project notes 2.1 - the biggest measured waste). sync8 builds its sync surface over the wide 0-5000 Hz span whatever slice asked, and every slice starts pass 1 from the same band (`dd8=dd8orig`), so twelve slices computed twelve identical surfaces. Now `sync8_wide` holds that part (the symbol spectra, the lag loops, the per-bin peaks and their baseline - everything that depends on the band, the DT window and the pass only), the driver calls `sync8_share` once per slicing on the band every slice starts from, its lag and peak loops in parallel, and each slice's pass-1 `sync8` reads the shared peaks (`red_sh` ... in ft8_mod1) and does only its selection; passes 2+ compute as before. Every unit but the retry (which has no sync8 pass) shares; the member units too, since `ens_perturb` writes `dd8orig` before the loop.

Two dead ends, measured, so nobody repeats them. (1) A single ‘sync8_wide‘ for both the shared and the per-slice calls with ‘!$omp ... if(lpar)‘ on its loops: the directive costs ~4 ms per call even with the ‘if‘ false - sixteen calls a pass put 250 ms of CPU back and the reply phase got 15 ms *slower*. (2) The same without directives, the loop bodies as contained procedures: still +20 % per call (an argument-passed band and host-associated arrays against the inline original), the reply phase 25 ms slower. So the per-slice path is the original code, verbatim and inline, and the split exists only for the shared call - two copies of the same text, with a comment saying why.

Result (24 on-air periods, light preset, background off, 12 threads, idle machine, two runs each; ‘JTDX_MEMO_STATS‘ sync8 wall summed over the slices, per period): pass 1 **0.37 -> 0.024 s**, passes 2-5 unchanged at 0.29-0.31 s each, the reply phase **0.842 / 0.852 -> 0.832 / 0.839 s** - about 1.3 % of reply time and 0.34 s of CPU per period; in the background every unit's first pass saves the same. The wall gain is half the 20 ms the project notes predicted: the hoisted spectra stay serial (their FFT buffers are equivalenced, which OpenMP cannot privatise) and the shared call itself costs 24 ms. As promised, not a decode moved: ‘ft8strict‘ and the four full-hour rows of ‘ft8repeat‘ byte-identical, ‘regress‘ 44/44, ‘bgabort‘ 14/14, ‘memcheck‘ clean; FT4 untouched (‘ft4strict‘, ‘ft4bench‘). **With the background on** (the same 24 periods, light preset ‘-B 1 -V 2‘, two runs each): reply phase 0.840 / 0.845 -> 0.824 / 0.821 s (-2.4 %), the background phase 3.43 / 3.44 -> 3.37 / 3.37 s, decodes 232 = 232 - the units' first passes share too. jtdxjt9 only, no params change; deployed 2026-09-02 as jtdxjt9 b20e73fa (rollback jtdxjt9.pre-item76-20260902). FT4's getcandidates4 (a fortieth the size) is left as it is: the FT8 lesson is that the sharing only pays where the surface is expensive.

77. **Back to 16 bit audio, both depths readable** (2026-09-03, the project notes). The fork had carried JTDX's `jtdx_32a` branch - 32 bit samples end to end - since 2026-08-22 (the project notes, `patches/`). Measured before deciding: the codec is 16/20/24 bit; every 32 bit recording's low 16 bits are PipeWire's resampling residue (about -90 dB, non-zero in 99.9 % of samples); the band noise floor is 83-89 LSB in 16 bit units on today's recordings and 16 on the lowest-gain one in the tree, against 0.29 LSB of quantisation noise; and **73 on-air periods decode identically at 16 bit** - FT8 25 periods 295 = 295 byte-identical, FT4 24 periods 145 = 145 with one line 1 Hz off. The 32 bit chain cost double shared memory, double disk per saved period (4.9 GB in `save/`), a Qt input format other backends

can refuse with no fallback, a live ABI mismatch on the 11025 Hz load path (`wav12_`), a 20-byte `fmt` struct in `getfile.cpp`, and four empirical display corrections. Reverted to upstream's layout with the per-file reverse patches (23 files apply clean) plus hand work in `mainwindow.cpp`: `read_wav_file` reads 16 bit natively and **contracts 32 bit files on the fly** (`wav32to16`, top 16 bits, chunked through a scratch buffer - the reverse of what the project notes described), both branches mono-only; `save_wave_file` writes 16 bit; the converter's 32 → 16 direction shares the same helper. **The decoder's float content is kept exactly**: `jt9a.f90` scales the shared buffer by 65536 on all four copies (id2 and the dd2 branch), the convention file mode already used for 16 bit files, so a power-of-two shift is the only difference and no recorded expectation moved. The simulators keep writing 32 bit (their noise is quantised at the output depth; re-scaling them would have moved every simulated-signal expectation for nothing). **Shared struct changed** (`d2` short, 13.69 MB): pair deploy, never mix with the 32 bit builds; `params_layout.sh` now pins the whole struct (size, `params` offset, 2-byte samples), not only the params block. Tests: `test_wav16` gained the contraction (extremes, floor at negative values, the 16 → 32 → 16 identity over all 65536 values, chunked = single call); the branch study's marker scan, now `test/verify_16bit.sh` on the routine list (64 markers, the d2/id2 width cross-check, and the fork's x65536 scale / 32→16 reader / 16-bit writer as must-be-present checks; proven to fail on each planted defect), passes; regress 43/44, ft8strict, ft4strict, ft8merge, ft4merge, ft4bg, ft4qso, ft4bench, bgabort, ft4hint, memcheck on the rebuilt build-vg, the preset harness - all green with nothing regenerated. The 44th row, `agcc_12t`, is unstable on the deployed decoder too (61/39/61 there, 61/61/39 here, expected 71): the project notes 2.10, not this change. **GUI, proven on** `onair_1h/260827_165800.wav`: the 32 bit file (ALLTXT: "32-bit wav file contracted to 16-bit on the fly"), its `make16.sh` twin in a fresh instance, and `jtdxjt9` file mode with the same recipe give the same 12 decodes, whole lines; a repeat of the 32 bit replay is identical; the S-meter reads on its 0-90 scale and the waterfall and cumulative curve look as upstream's. Built as the pair jtdx 35bc9202 / jtdxjt9 82086907 and **deployed 2026-09-03** (rollback pair ~/jtdx-prefix/bin/*.pre-item77-20260903 = jtdx 37c9485b / jtdxjt9 b20e73fa, matching halves only - the last 32 bit pair). On-air check after the deploy: the first saved period is 16 bit; the TX drive is the operator's to confirm on the first transmission (the modulator's full scale maps to the same analogue level, but its ramp constants changed).

78. **FT4's TX background enabled as FT8's is: one explicit switch** (2026-09-04, the project notes 2.13, the project notes "The interface, re-checked"). **PARAMS BLOCK 360 → 364 BYTES** (`nft4bgeffort`, appended after `ndecreq` so no earlier offset moves), pair deploy. Until now the FT4 phase ran only when the background's member target exceeded the RX member count - its unit WAS the members - so the background's extras (deep OSD, alternate pass, residual, its sensitivity and QSO-frequency level) vanished with the last member, "most at reply time" and "max effort" could not have a background at all (why max effort sat below recommended), raising RX members silently switched the background off, the thread ladder could make the switch inert on a small machine, and the GUI duplicated the decoder's rule to know whether a `<BackgroundFinished>` line would come (the 2026-09-04 separator defect lived in that duplicate). Now the GUI's "TX background decoding" switch is sent as `nft4bgeffort` exactly as FT8's `nft8bgeffort` is; `jt9a` runs the phase when it is set, whatever the target; the members above the RX count run, and with none left the phase's extras run alone (~1 s of idle time on 12 threads); a status line always follows; the spacer flag reads the switch; the preset matcher compares the switch, then the members the phase runs, then the phase's fields; the two menu strings lost their "must exceed" clauses. File mode: `-B 1` is the FT4 switch as it is FT8's, `-V` the target, and `-V` alone no longer implies a background (`ft4strict`, `memcheck`, `ft4qso` pass `-B 1`; `ft4bg` has "off" and "zero" cases and checks the field in the params dump; `test_preset` pins the rule; the project notes updated). One FT4-only condition stays: the unit needs more

than one thread, since it runs over the slice grid. Both trees, all guards green on the pair (battery below); built as jtdx 091a3ea8 / jtdxjt9 263c8b67 (public e2743de9 / b41265c5), **NOT deployed** - the 360-byte pair ff0f41da/b94ab4ce + 04f5e0e2 stays on air until told.

79. **The FT4 presets re-swept over all sixteen recipe fields, and the two reply-time tiers re-pinned** (2026-09-04, the project notes 2.9; the project notes has the tables, the harness and the raw TSV sit beside it). 24 recipes on seven noise realisations of the crowded file at 0-5000 Hz and on the 240-period night hour at 100-3100 Hz. Findings: the narrower range costs nothing on the night hour (1748 / 1888 / 1858 as at 0-5000); the zero-member background (item 78) is what the reply-time tiers were missing - six members in the period plus the background's extras gives **1886 total** / **1845 at reply time at 0.55 s** against the old most-at-reply-time's 1842 / 1842 at the same cost; everything in both phases gives **1891** / **1878 at 3.1 s** (max effort above recommended at last; it was 1858); recommended stays (its candidates: RX low thresholds +6/+21 on the night but -8 at reply time on the crowded band; background low thresholds +8 for +29 % idle CPU; RX alternate pass +12 at reply time, -2 crowded - all operator options); reply-time members or OSD on top of recommended lower the total; best power stays plain bg 3 (the best gain per idle second). Zero false on the night hour in every recipe; six RX members produce one false in seven crowded runs, as before. New recipes: "most at reply time" = members auto + background target auto with deep OSD, alternate pass and residual (bgon, no member left); "max effort" = 6 members + deep OSD + alternate pass + low thresholds at reply time, the extras + low thresholds in the background. Labels, `test_preset` and `ft4bench` (two new rows, 77/0) updated; jtdx-side only on top of item 78's pair. Not re-measured: the day hour (by choice of data); the QSO-frequency levels (inert without a QSO in progress, item 75's simulated QSO measured them).

80. **FT8's budget features ported to FT4: RX budget auto, the learned-cost background clock, the two-period window, a per-mode margin** (2026-09-04, the project notes 2.13; operator's rule: every preset's RX phase stays near the 1.36 s deadline and the extra effort goes to the TX background, max effort's until 0.5 s before the window ends). No params-block change - the field values already exist (`nft4ensemble = -2`, `nrxbudget`, `nbgbudget`, `nbgmargin`).

- **RX budget auto** (`FT4_ENSEMBLE_BUDGET`, menu "budget auto" in the FT4 RX ensemble submenu, ini `FT4EnsembleEffort=-2`, file mode `-M budget` with `-l TENTHS`). FT8's P8 in FT4's shape: FT4's members are extra sweeps per candidate inside the slice loop, not separable units, so the count is decided BEFORE the decode from the learned cost of the whole RX phase at each member count (`ft4rxcost(0:6)`, ft4_mod1): the largest count whose cost fits the budget (`FT4RXBudget`, 13 tenths = 1.3 s by default), the first that does not fit decayed by 10 % so it is tried again; a run blends 50/50 at its count and seeds every unlearned count by the built-in ratio (0.30 s + 0.12 s a member at 12 threads, the crowded file's figures, thread-scaled); learning happens in every mode so budget auto starts warm. The count that ran is reported in `<rxm>` (the GUI's "RX budget auto: n member(s)" line now covers FT4) and the background phase continues from it (`nft4rxrun`) instead of from the setting. On this machine all six fit on both datasets; the value is on fewer threads and crowded bands, where the static ladder cannot see the band.
- **The background clock on FT8's gate**: each slicing (first, second, residual - cost kind 7 in the shared `bgcost` table) is started only when its learned cost fits before the deadline, `cost_learn` / `cost_decay` exactly as FT8's units; before, the first slicing always ran and the later ones were gated by the previous slicing's time. Caught by `ft4bg`'s abort cases on the first build: the estimate was thread-scaled by `numthreads`, which is FT8's variable and stays 0 on the FT4 path, so every slicing was priced at 12 s and skipped under the lock rules; kind 7 now scales by FT4's own thread count.

- **The P7 two-period window** reaches FT4 (`mainwindow.cpp`: the budget and the own-TX-period skip are computed for whichever mode has its background switch on); `test_budget` pins the 7.5 s figures (75 / 150 tenths).
- **FT4's own budget and margin** (`FT4RXBudget` 13, `FT4BgMargin` 10, sent as `nrxbudget` / `nbgmargin` in FT4 mode); the max effort preset sets the margin to 5 (0.5 s) - not a recipe field, since it changes when the phase stops, not what it decodes.
- **Max effort re-pinned on the rule**: budget auto members + the alternate pass + low thresholds at reply time, everything in the background (deep OSD, alternate pass, residual, low thresholds); RX deep OSD is out of every preset (2.5 s on a crowded band with six members). Measured (the project notes, addendum): the RX alternate pass with six members is out (1.5 s mean on the crowded seeds, 2.8 s worst on the night, +11 at reply time, +0 total); the RX low thresholds fit (+14 at reply time, +0.11 s); the background low thresholds add +7 in idle time. The recipe as `-M budget -l 13` on this build: **night 1894 total** / **1859 at reply time** / **0.69 s mean** / **1.42 s worst** (+8.4 % / +6.4 % over the default), six members admitted on every period; crowded seeds 506 / 500 at 1.43 s with six admitted on each single-period run - the learning cannot carry between separate processes there, on air it does and would trim to five. Tests: `test_preset` (budget auto stored, passed through, the matcher compares the target itself under budget auto), `test_budget`, `ft4bg`'s `rxbudget` case (`-l 1` admits no member and equals `-M 0` on every period, `-l 600` admits six and equals `-M 6`), the strict rows, `ft4bench`, `ft4qso`, `ft4merge`, `params_layout` (364 bytes unchanged) green; both trees. The harness links `build/`'s objects, so it must be rebuilt after a header change or the linker keeps the stale inline copies (bitten here: two false failures until `make jtdx`). **Review (2026-09-05, early):** one defect - file mode's `nrxbudget` starts at 27 (FT8's 2.7 s), so `-M budget` without `-l` ran FT4 at a 2.7 s budget where the GUI sends 1.3 s; the default is per mode now (13 tenths for FT4 unless `-l` is given; the measurements above all passed `-l 13`, so no number moves). Checked and sound: the background call returns before the RX block, so its time never enters the RX cost table; the P7 window and the own-TX-period skip are live in FT4 (`m_txPeriod` is set on every TX start whatever the mode, and the 5 s trigger offset stays inside a 7.5 s period); `-2` reaches only the params fill and the matcher; the cost table is shared across recipes, so a preset change re-learns within a few periods, as FT8's does. Coverage added: `memcheck`'s FT4 QSO case now runs `-M budget -l 13`. Not yet covered: the learned count actually dropping on a crowded band (needs a chain of crowded periods; the seven seeds are single-period runs where learning cannot carry).

81. The decode label shows an X after a TX background that was cut short (2026-09-04, asked for from the air). The label reads "/(D+N=S) |" while the background runs and "/(D+N=S)" when it finished; now "/(D+N=S) X" (the X on the background's red) when the phase did not finish because the next period's decode started - `<abort>1` from the decoder - and stays until a background completes again: the visual clue that the machine load or the effort level wants a look. The GUI's own aborts (a band or mode change, `abortTxBackground`) are not the load's fault and do not set it. The per-line | markers cannot say this, since the decodes print before the phase is cut. `decodelabel.h`, both trees; `test_label` pins the plain and the rich-text forms; the ALL.TXT line already said "cut short by the next decode".

82. The public numbers re-measured on the deployed pair, with the real stock decoders in the table (2026-09-05, the project notes 2.14). Every experiment behind the published figures was rerun on jtdxjt9 8a3e17ee (items 78-81), one row at a time on an idle machine, and the tables were rewritten from the outputs (`analyse.py -update-docs`: DECODE_RECIPE_PLAN 13 / 13a, FT8_DECODER 5, the `preset_onair_*.tsv` files):

- **FT8, the on-air hour, 17 configurations.** Every JTDX_contest preset reproduced within

half a percent of the 2026-08-28 run (light 6091, pipeline ensemble 6194, full 6208, run 6234; WSJT-X's own multi-thread decoder moved 5188 → 5171 between two runs of the same binary, so about one percent is the noise floor). New rows: the real stock v2.2.159 through `jtdx_stock_filemode/` - as shipped (9 cycles, sensitivity 2, ALLCALL7.TXT) 4802 at one thread and 4783 at its GUI's 12 threads; without the known-call file 4869 / 4866 (the lookup rejects 67 real decodes); at its deepest (SWL mode on top) 4767 / 4793, i.e. SWL mode loses on this band. The public baseline (operator's call, later the same morning) is stock at its honest best, the known-call lookup off: **6208 is +27.5 %** (the recommended preset +25.1 %); +29.3 % against stock as shipped, +18.3 % against the same decoder at JTDX's own recipe (5249). The "classical" stand-in read 5564 on the first pass - the runner's `read` had collapsed an empty field and dropped its `JTDX_HINT_DEPTH=1` (so the row ran at depth 4); both runners fixed, the row rerun.

- **FT4, both hours, 10 configurations** (`test/decode/onair_ft4/`, new): stock as shipped 1356 on the night hour (single-threaded, 1.35 s a period), 1525 without ALLCALL7.TXT (the lookup rejects 169 there), the fork without its hint memory 1552, default 1737, best power 1822, recommended 1881 (0.10 s), most at reply time 1878 (0.48 s), max effort 1888 (0.59 s), WSJT-X 3.0.2 1561, WSJT-X improved 1688; day hour 494 / 507 / 516 / 543 / 553 / 561 / 560 / 560 / 515 / 536. The tiers run with `JTDX_STOPHINT=1` as the FT8 experiment does, hence 1881 here against 1888 in the item 79 sweep. The member ladder for the explainer's chart, night hour: 1748, 1760, 1791, 1811, 1826, 1840, 1842, alternate pass 1853; without the hint memory 1565.
- **Documents refreshed**: README (both trees, the engine table on the stock baseline, the preset tiers), the deck (headline bars with the stock rows, every FT8 figure, the FT4 slide), the explainer (engine table, headline stats, benefit cards, the FT4 chart and captions, six languages), the paper (abstract, tables 1-3, figure 5, conclusion), FT8_DECODER.pdf and the project notes (the auto tables and the new FT4 sections). The per-kind attribution of the FT4 ensemble's gains stays the 2026-08-30 measurement and is labelled so.
- **Every classical-based percentage moved to the stock baseline** (2026-09-05, operator's call): the preset tiers in the README, the FT8 preset menu labels (best power +11.3 %, best value +16.6 %, best medium effort +25.1 %, best results +27.2 %, max effort +27.5 %, most results +28.0 % - before: +3.1 / +8.0 / +15.7 / +18.2 / +18.3 / +18.5 % against the classical recipe), the deck's tier slide (axis rescaled to +30 %, the knee now 90 % of the maximum gain for 40 % of the CPU) and its callsign-check slide (`extra_calls.py` rerun with `JTDX_EXTRA_BASE=stock_mt_noallcall`: 624 / 860 / 1229 / 1339 / 1371 extras, 99.8 / 99.5 / 99.1 / 98.1 / 98.5 % with every call known; 38 doubtful, 25 at -23 dB, 29 from the TX background; no RTTY serials left), and the FT8 auto tables (`onair_1h/presets.tsv` now leads with `stock_mt_noallcall`, so the "not in" / "only" columns count against stock). Left on the classical recipe by design: the explainer's benefit cards ("Every approach against the classic JTDX recipe, by mode" - the per-feature A/B deltas, renamed; the five tables the script built once at load - ALT, ATTR, ENGINE, BEN, GAP - are functions now, so they re-translate on a language change instead of keeping the first language's text), the paper's three "at JTDX's own recipe +18.3 %" clauses (restored on the operator's call), the hint-depth chart, and section 13's SNR-distribution prose (a note there gives the +7.8 % conversion).
- **The Ensemble preset left the FT8 presets submenu** (2026-09-05, operator's call): its QAction, slot and action-group wiring removed in both trees; `DecodePreset::Ensemble`, its recipe, the lamp letter `E`, the file-mode path and `test_preset`'s coverage stay, and the lamp's tooltip names it when the controls match. Built (bin/jtdx f3fb89e0, public tree 689e8616), not deployed.

83. The ALLCALL7.TXT parse gated behind the lookup it feeds, and the Windows FT4

crash triaged from Linux (2026-09-05, reported from a Windows build of the public tree: the decoder died in FT4 mode, and every start complained "ALLCALL7.TXT is too short or broken?").

- **The message.** `cwfilter` opened ALLCALL7.TXT with `status='unknown'`, which CREATES an empty file wherever the share directory has none - an install that never received the file then reported its own empty copy as "too short or broken?" on every start. The open is now `status='old'` with `iostat`: a missing file is never created, the parse is skipped, every count stays at zero, and the existing check reports it once and sets `ldbvalid` false (`searchcalls` then accepts every callsign, JTDX's behaviour without the file). The parse itself stays UNCONDITIONAL. A first version of this item gated it behind `chkflscall`'s switch on the claim that the tables had no other reader; the review pass showed the claim false - `ft8b.f90:1737` (the AP type 35/36 acceptance of a /R callsign) and `chkfalse` (the JT65 false-decode filter, from `extract` and `filtersstd`) call `searchcalls` directly, and with empty tables and `ldbvalid` true they would have rejected every decode they were asked about. That version (jtdxjt9 f209de8d, then fa6a2c44) was deployed for about forty minutes on 2026-09-05 and rolled back to dc7153e9 the moment the review reported; the FT8 strict guard had not caught it because its 240 periods hold no /R decode on that path. So the Windows message is legitimate: that install has no ALLCALL7.TXT in the directory the GUI passes with `-r` (`contrib/CallDB/ALLCALL7.TXT` is installed by `cmake -install`; a build tree run in place has none) - and it very probably has an empty one created by the old open, to be deleted before the real file goes in. Both trees, ten changed lines in `cwfilter.f90`; `chkflscall.f90` and `ft8_mod1.f90` back to their previous text. Corrected build jtdxjt9 5b6d3d72 (public tree 31885cb8): FT4 and FT8 strict guards byte-identical; DEPLOYED 2026-09-05 (jtdxjt9 5b6d3d72 alone, jtdx unchanged, rollback jtdxjt9.pre-item83-20260905 = dc7153e9), AppImage 126b2023 from the public build, extraction-checked.
- **Review follow-up, not acted on:** `maskincallthr` has stride 8 but `ft8b.f90:2002` tests `nindex.lt.maskincallthr(nthr+1)`, so a slice stores 7 incoming calls, not 8; the inherited JTDX mask wasted the same one slot, the dense stride makes it an eighth of the capacity. Either `.le.` or a documented 7 - to be measured, since it can change decodes.
- **The crash, what Linux can say.** The saved period that reproduces it on Windows (`jtdx_crash/260905_175508.wav`) is 72576 samples of which 70290 are exactly zero - a quarter second of signal, then silence. Under `-fcheck=all` (build-chk), under valgrind at 1 and at 12 threads, on that file and on a period of pure zeros, the Linux decoder is clean: no out-of-bounds access, no uninitialised read reaching a branch, 96-106 candidates, no decode. The one way Linux reproduces a crash is a thread stack under 5 MB: the plain FT4 recipe at 12 threads dies in `getcandidates4` (its 630 KB `s(NH1,NHSYM)` on top of `ft4_decode`'s frame) at 4.6 MB and runs at 5 MB; FT8 needs between 6 and 8 MB. gfortran under `-fopenmp` puts every local array on the stack, so these are the per-thread requirements, and Windows threads get the executable's reserve, which CMakeLists sets to 16 MB for `jtdxjt9` - enough if that flag reached the Windows link. What to check on the Windows side, in order: the decoder's actual stack reserve (`objdump -p jtdxjt9.exe | findstr SizeOfStackReserve`, expected 0x1000000), whether a full-length FT4 period saved on Linux crashes the same way (if not, the fault is data-dependent and the silent file is the key), whether FT8 file mode works at the same thread count, and the crash text itself (Fortran runtime error, access violation, or a silent exit).
- **The Windows trace, later the same day.** gdb attached to the GUI-started decoder: status c0000374 (heap corruption detected) raised inside `RtlAllocateHeap` under FFTW's planner, `sfftw_plan_dft_r2c_1d` from `four2a` from `ft4_downsample` from `ft4b`, on an OpenMP worker of the first FT4 period after a switch from FT8 - i.e. the damage was done earlier and the first heap request of the FT4 period found it. Facts from the mounted Windows

disk: the build is the public tree at e46db97 (line endings apart, identical), toolchain x86_64-posix-seh **GCC 8.1.0** (Qt's mingw810_64), FFTW 3.3.10, stack reserve 0x1000000 as linked, `OMP_STACKSIZE=24M` passed by the GUI, and 116 `__emutls` symbols: on that toolchain every `threadprivate` array (`dd4`, `dd8`, the downsamplers' `cxx`, the sync templates, ...) is a per-thread HEAP block, where Linux keeps them in static TLS. File mode on the same Windows machine decodes the crash period cleanly; the GUI dies on every FT4 period and the watchdog restarts the decoder each time (ALL.TXT "Decoding forcibly recovered", 37 s apart). On Linux the GUI's sequence - one FT8 period, then FT4 periods, in one process at 12 threads, with the partial-interval shuffle and AGC compensation - is clean under `-fcheck=all` and under valgrind (`JTDX_FILE_MODES=844`, `JTDX_NDELAY`, the two hooks added for this). GCC here rejects `-femulated-tls`, so the emulated-TLS layout cannot be mimicked on Linux. Next on the Windows side: threads = 1 in the GUI (no worker, no per-thread emutls block: if it survives, the fault is in the threaded path), then a decoder rebuilt with `-fcheck=all -fbacktrace` at `-O1` (a runtime error names the line; a crash that vanishes at -O1 points at the 2018 optimiser), then a full page heap if both are inconclusive.

- **ROOT CAUSE, found on the Windows side the same night** (report and gdb scripts in jtdx_crash/ on the Windows disk, the project notes): `ft4b.f90`'s first-call block filled the 64-element columns of its twiddle table `ctwk2(2*NSS,-16:16)` through `twkfreq1(ctwk,0,2*NSS,2*NSS,...)`, and twkfreq1's loop is `do i=0,npts` - inclusive - so each call stored 65 elements. Columns -16..15 spill into the next column's first element, which the next iteration overwrites; the last column spills 8 bytes past the end of the array. Inherited from JTDX, where ctwk2 is plain static data and the spill landed in whatever followed it. The fork made ctwk2 `threadprivate` for the slice loop; on Linux that puts it in static TLS, where `readelf` shows the 8 bytes after it are alignment padding before `apbits` - so no Linux decode was ever touched, which is why every guard stayed byte-identical and why -fcheck could not see it either (an explicit-shape dummy sized by the caller's own arguments). Under MinGW's emulated TLS the array is an exactly sized heap block and the spill overwrote the next block's header. Fix: last index `2*NSS-1` at the call site - 64 stores, the same cumulative phasor, bit-identical tables (ft4b.f90); twkfreq1.f90 carries a header documenting its real contract and the deeper fix not applied (loop over nbot:ntop, with ft8b's call changed to match - the ft8b call is in bounds today). The file-mode hooks were reviewed on that side too: JTDX_FILE_MODES switches a loop-local copy, JTDX_NDELAY applies to one file (JTDX_NDELAY_FILE=k picks it), an over-long argument is skipped with a message. Both trees; README carries a Windows paragraph with the toolchain fact and the gdb recipe (_NO_DEBUG_HEAP=1, the CRT's _heapchk). Built as jtdxjt9 9960d157 (public 9d63a3a9); the strict guards must stay byte-identical - FT4 and FT8 strict guards OK. DEPLOYED 2026-09-05 (jtdxjt9 9960d157 alone, jtdx unchanged, rollback jtdxjt9.pre-twk-20260905 = 5b6d3d72), AppImage b0d9e422 from the public build, extraction-checked, copied to public/download.

- **Not a fork change: the 72576/73728 pair.** The Windows-side review read `NMAX=73728` against a save path of `21*3456` as something this fork introduced; both values are JTDX 2.2.159's own (`jtdx_contest_org`: `ft4_params.f90:13`, `jt9a.f90:70`, `mainwindow.cpp:1954`). The decoder reads 1152 samples past what the GUI saves; on a file they are zero-filled by `read_wav_file`, on the air they are the ring buffer's content at those positions. Left as JTDX has it, noted here so it is not re-discovered.

84. The decode label written tight, so the trailing marker survives Windows (2026-09-06, asked for from the air). The count read `Avg=0.19 Lag=+0.71/(31+0=31) |`; it now reads `Avg=0.19Lag=+0.71/31+0=31|`. Three things went: the space before `Lag=` (`m_decodeLabelMid`), the brackets around the RX+TX sum, and the space before the trailing marker - four characters in the background states, two in the others. The point is the marker at the end (`-` decoding, `|` background running, `X` background cut short): on Windows the label is clipped

at the right edge well before it is on this machine, and the marker - the one part that says what the decoder is doing right now - was the first thing to go. Nothing is harder to read for it: D, N and S already carry their own tints (blue / red / green) and the average and the lag theirs (lavender / amber), so the colours separate the fields that the spaces and brackets used to. `decodelabel.h` (both forms), `mainwindow.cpp`, `mainwindow.h`, both trees; `test_label` pins every state and now also asserts that no state contains a space or a bracket at all.

85. **The Band Activity chip stops eating the header row** (2026-09-06, asked for from the air, the sibling of item 84). The state chip at the right of the Band Activity header (`label_6`: "check time" when the average DT is off, "lost audio N", otherwise the pane's own name) was `Expanding` in a 9:2 layout, so it took about 18 % of the row at every window width - and what it took came off the end of the column header, which is where the decode count and its marker live. It now hugs its own text (`Maximum`, the stretch moved to the header, whose 1000 px cap is gone), and the idle word is "Band" rather than "Band Activity". Measured on the real `mainwindow.ui` at a 381 / 481 / 781 px pane, the header gains **33** / **51** / **106 px** - four to fifteen more characters of Avg, Lag and the count. A warning still gets its full width, since the chip grows to whatever text it holds ("check time" 77 px, "lost audio 12" 91 px), and it can still shrink below that when the row cannot hold both. `mainwindow.ui` and the seven `setText` sites in `mainwindow.cpp`, both trees; `test_headerrow` pins the split, the warning widths and the shrink.

Part II — WW Digi contest support

WW Digi contest support - all enhancements and fixes

The companion inventory to `IMPROVEMENTS_WW_DIGI_2026.pdf`. Each item names the plan document with the full detail. Everything is implemented, unit-tested and replay-verified in the GUI; **nothing has been used on the air yet** (contest: Aug 29-30, 2026).

The contest mode itself (the project notes)

1. **Settings** → **Contest tab** - a radio group selecting the contest; the full WSJT-X `SpecialOperatingActivit` enum is declared so stored values keep their meaning, with NONE and WW Digi offered and the rest disabled placeholders. The old `Common/WWDigiContest` key migrates on first read. The tab is translated.
2. **The settings interlock** - selecting the contest parks twenty-five settings (grid highlighting on / per band, autolog, clear-after-log, distance to comments, the unscored DXCC/zone/prefix tiers and their beeps off, ...), forces and greys them, and hands them back on leaving. The parked set persists, so a restart mid-contest still knows what it owes.
3. **The window title names the running contest** (`JTDX_contest - WW-Digi - ...` since 2026-08-25; the application name behind the settings and data directories stays `JTDX`), so an instance left running for days is identifiable from the task bar; the shared-memory ID and startup line keep their old format.
4. **Receive side** - the message that did not work: `MYCALL HISCALL R GRID` now classifies as `RRREPORT` (answering with RR73) instead of dead-ending; `report()` returns the grid as the exchange (contest-gated, all four call sites pass the live flag); a QSO whose only decode is `R GRID` still logs the gridsquare.
5. **Transmit side** - Tx3 is built as `HISCALL MYCALL R GRID`, activating the dormant contest branch; the published 4-message sequence (CQ WW / grid / R grid / RR73) runs end to end.
6. **What reaches the log** - the rst fields carry the decoded SNR (never grids), the grids go to `gridsquare`/`my_gridsquare`, and the comment reads `WW-Digi 2026 - 10684 km`, the year taken from the QSO, not the clock.
7. **No usable station grid = no transmission.** The grid is the exchange, so with an invalid grid every Tx message is cleared and Enable Tx switches off, with the reason in ALL.TXT - instead of transmitting a placeholder that would reach the other operator's log as a genuine exchange (placeholders were tried and proven unusable).
8. **Rule XI.12 respected** - autoselect answers stations calling us but never initiates a call while a contest runs; search and pounce stays a human act.
9. **Sequencer fixes** - `RCALL` replies with Tx3 when his grid is known; the grid exchange is kept out of the report spin box; `CQ WW` **normalised to an undirected CQ** - as a directive it matched no continent or prefix, so autoselect silently skipped every station calling `CQ WW`.
10. **Five bugs found and fixed during implementation**: own `R GRID` transmission was recorded as `S73` (marking the QSO finished the moment Tx3 went out); `RRREPORT` was gated on having sent `SREPORT` (never true in WW Digi); the closing 73 was discarded after a double-click (empty `s_rep` blocked the terminal states - both report fields now refill from the decoded

SNR); Tx3 painted pink (the old 72-bit packer called it free text); and a half-cleared message set halting TX with a misleading error.

Scoring and multipliers

11. **Contest points** (the project notes) - `1 + km/3000` per QSO, confirmed against the published rules; shown in the message window and written to the ADIF comment. Distance from 4-character grid centers. A message without a grid (distance unknown) shows `-` in the tag's width instead of nothing, so the country column stays aligned (2026-08-28, `contest_points_tag()`).
12. **The priority ladder** - contest points rank above the unscored tiers, so the highlighting and autoselect ranking agree with what actually scores (a "new DXCC" would otherwise outrank a 7-point station).
13. **New grid FIELD highlighting** (the project notes) - the multiplier is the 2-character Maidenhead field, so in contest mode the "new grid" machinery (colours, per-band, beep, preview) tracks fields instead of 4-character squares; a new field outranks new DXCC or zone.
14. **A separate contest log** (the project notes) - every contest QSO is written to a second ADIF file, and while the contest runs all inference (worked-before, multipliers, highlighting, autoselect ranking, the score display) comes from that file. A correctness fix, not bookkeeping: multipliers reset every contest, and hashes built from the lifetime log stop highlighting exactly the rare fields worked in an earlier year. Includes the live score display.

The main window during a contest (the project notes)

15. **Eight main-window controls taken over**: Hound off, single shot off, AutoTx on, Max distance and report priority off (the tie-break among equal-value callers stays SNR), Skip Tx1 on, RRR off, AutoFilter off, mode restricted to FT8/FT4. Operator preferences (AutoTx, Skip Tx1, RRR, mode) are parked and restored on leaving; Hound / single shot / AutoFilter are left off deliberately. Both widgets of every paired control are locked (the `1 QSO` button was found still clickable).
16. **Contest frequencies** (the project notes) - a contest keeps its own working-frequency list, edited on the Contest tab with the same table the Frequencies tab uses; the band selector offers it while the contest runs.

Supporting work

18. **16-bit wav support** (the project notes) - File → Open expands 16-bit files to 32-bit in memory (WSJT-X and most tools write 16-bit; JTDX decoded them as noise); plus File → Convert bit depth (32/16) and a pre-existing buffer overflow fixed in passing. Verified: 16-bit twins decode identically to the 32-bit originals.
19. **Simulator repairs** (independent of the contest) - `watterson.f90` had lost a parameter while all three FT simulators still passed it, shifting every argument one place: any simulation with fading or delay produced audio no decoder could read. Restored. The FT simulators now write the 32-bit wav format JTDX replays (they produced files JTDX itself could not open) and clamp before int32 overflow.
21. **Report messages rejected in contest mode** (2026-08-29) - the WW Digi exchange is the grid, so a decoded line ending in a signal report (`K1ABC W9XYZ -10`, `... R-10`) is either a station outside the contest or a false decode (an a-priori codeword's random g15 field is a report as often as a grid, and the report form has no `chkgrid` consistency check

to fail). `readFromStdout()` drops such lines while a contest runs - before ALL.TXT, the windows, the UDP clients and the sequencer (`contestreport.h`, `test_contestreport`); with the debug ALL.TXT on they are logged as `contest: report message rejected`. RRR / RR73 / 73, grids, CQs, hashed and free-text messages pass.

22. **Status bar in a contest** (2026-08-29) - the mode QSO counter (`FT8 n`) counts the contest log, i.e. the QSOs of this contest, instead of the everyday log, and the date and "Logd" (last logged) labels are hidden while a contest runs; both follow `updateContestScore()`, so they switch over with the contest and refresh with every contest QSO.

23. **One multiplier colour** (2026-08-29) - in a contest a field that is new on this band is painted in the *new grid* colour like a field never worked, instead of the *new grid band* colour: both are multipliers (WW Digi counts fields per band), and two colours for one fact made a fresh band look like everyday operation. Autoselect ranks them equal as well (both 48/49 - one multiplier is one multiplier), so between two multipliers the SNR tie-break decides; 46/47 is no longer used in a contest.

24. **New call, the contest dupe check** (2026-08-29) - the new call triple joins the interlock: parked on the way in, forced *new call* on, *per band* on, *per band and mode* off (a station counts once per band whatever the mode), restored on the way out, greyed meanwhile; and in a contest "new call on this band" is painted in the *new call* colour rather than the band colour - it is the dupe rule itself, so one colour says "not worked here yet". 28 settings parked now; `test_interlock` covers twelve.

25. **Double click on a caller answers with R GRID** (2026-08-29, seen on air). Calling CQ, three stations answer; the first QSO finishes, the auto-sequencer picks the second caller and correctly sends `HISCALL MYCALL R MYGRID`. The operator then double clicks that caller's line (`MYCALL HISCALL GRID`) and the transmission changes to `HISCALL MYCALL MYGRID` - Tx2, the bare exchange, as if I had initiated the call. Cause: `processMessage` kept JTDX's plain-FT8 rule "my call in the message and a grid at the end → Tx2 (send my report)"; in the contest Tx2 is the grid without R (item 3) and his grid is already his exchange, so the answer must be Tx3 `R GRID` - what the auto-sequencer's `RCALL` branch already does. The decision now lives in `contestreply.h` (`reply_tx_to_me`): grid at the end → Tx3 in WW Digi, Tx2 otherwise; RRR/RR73/73 → Tx5, `R...` → Tx4, a report → Tx3, nothing → Tx2, all as before. `test_reply` pins the eighteen cases; confirmed on air the same evening (double click on a caller → Tx3 `R GRID` selected and sent).

25b. **The report filter was inert in FT4** (2026-08-29, found reviewing item 26). Item 21 and everything built on it read the message out of the decoder line by looking for FT8's `" ~ "` separator. FT4 prints `":" - 203400 1 0.2 300 : CE3TSK LU9XJR -03` - so the line never matched, and the filter has been doing nothing in the mode most of WW Digi is worked in. The evening's ALL.TXT proves it: nine report messages addressed to us survived into the log, from six stations, and the 20:34 sequence shows the cost - `CE3TSK LU9XJR -03`, we answer `LU9XJR CE3TSK R FF46`, he repeats the report, we repeat the exchange, and after two wasted transmit periods he gives up and calls CQ. `contest_message_text` now strips the decoder prefix by its shape (optional date, time, SNR, DT, frequency, any single separator character), so every mode is covered; the tilde remains as a fallback. Pinned in `test_contestreport` and `test_ignore` with those real lines.

26. **A station that answers with a report ends the QSO** (2026-08-29, asked for from the air). The WW Digi exchange is the 4 character grid. A station running ordinary FT8/FT4 answers our `HISCALL CE3TSK FF46` with `CE3TSK HISCALL -10` or `CE3TSK HISCALL R-10`: he waits for a report we never send, we wait for a grid he never sends, and the QSO cannot complete. Item 21 already dropped such lines before the windows and the sequencer saw them, which left the QSO hanging until it timed out - and the moment he called again the autoselect picked him first, because a station calling us outranks everything. Now the line

is inspected before it is dropped: if the report is addressed to us (`contest_report_to_me`, `contestreport.h`), he leaves the QSO history whatever he was to us - the station in the DX field, a caller we had not answered yet, or one never heard before - and if he was the station we were working the QSO is dropped as well (`clearDX`). He is then skipped for five minutes. The skip is enforced by keeping him out of the QSO history rather than by discarding him after he has been chosen: `autoseq()` returns one station per pass and is disarmed by that pick, so throwing the pick away would cost the whole period, and the caller behind him would wait. `process_Auto` therefore purges every station still inside its window from the history before it looks at anything (`ContestIgnore::calls`), which also covers the station who calls again with a proper grid inside those five minutes. It purges with a new `QsoHistory::forget()`, not `remove()`: `remove()` erases the operator's own blacklist entry as well (Ctrl + right click on Clear DX), and a skip that fires by itself every pass must not quietly undo a deliberate action - `test_wwdigi` covers that too. A caller whose message to us carries a report instead of a grid never enters the waiting list in the first place: item 21 drops the line before the history sees it. A double click on a report message is a no-op as well - the first thing `processMessage` does in contest mode, before it touches the Tx minute or anything else. Starting such a QSO by hand would only repeat the deadlock, the click is far likelier to be a slip than a decision, and it means an accidental click cannot clear that station's five minute skip either; a click on any other line of his - the CQ, the grid - still works and still overrules the skip, which is the operator's way in.

```
**What the operator sees** (added 2026-08-30, after the first on-air report of it): the rule
used to fire silently. The offending line is dropped by item 21, so from the operator's chair
a station simply stopped being answered and the newest thing on screen was his previous,
perfectly ordinary message with his grid - no way to tell whether the decoder had died, the
station had gone, or a rule had fired. Now one amber line goes into both decode windows and
into ALL.TXT unconditionally: *"KD6WW answered with a report - not the contest exchange, QSO
dropped, skipped 5 min"*, or *"... is sending reports, not the grid exchange - skipped 5 min"*
for a station that was only calling. Once per station, so a repeating caller cannot fill the
window (‘DisplayText::displayContestNotice‘).

**And the rule was too wide** (narrowed 2026-08-30, after it cost a multiplier). 3D2USU,
Fiji, grid RH82 - a new field - called ‘CQ 3D2USU RH82‘ on 14.080 FT4 at 06:39, we answered,
and he acknowledged with ‘CE3TSK 3D2USU +05‘ because his software runs reports. This filter
swallowed the acknowledgement, so the sequencer never advanced past Tx2; the rule then
dropped him and skipped him five minutes, silently, and we went back to CQ after a single
call. KT2R and HK3G completed with him in those same minutes. **His grid was already ours,
from his CQ** - the exchange existed and the QSO was worth finishing; item 26's deadlock is
the case where we hold *no* exchange from him. So a report from the station we are working
now passes through to the ordinary path when his grid is already known, where it advances
the QSO exactly as his ‘R GRID‘ would, with a notice saying so. Everything else drops and
skips as before. Where we go next is the sequencer's
decision, not CQ by default: ‘clearDX‘ leaves us calling CQ as the fallback, and with the
DX field empty ‘process_Auto‘'s autoselect runs again. It runs at ‘<DecodeFinished>‘ with
all of the period's decodes in hand, and the abort clears ‘m_processAuto_done‘ so it runs
even when an earlier line of the same period had already triggered it: the decision is
taken in this period, not the next. A double click on him overrules the skip, and leaving
the contest empties the list. Reports
between two other stations are still only dropped, never acted on. The skip list is
‘contestignore.h‘, ‘test_ignore‘ pins its 33 cases including the second report restarting
the five minutes and a clock that jumps backwards never releasing a station early, and
‘test_wwdigi‘ scenario F drives it through the shipping ‘QsoHistory‘: the aborted station is
not picked again, a station who sends a report is passed over for the next one in the same
pass, the operator's blacklist survives the purge, and without the abort the same history
sits at ‘SRREPORT‘ repeating the exchange - fourteen checks, and they call ‘forget()‘ exactly
where the shipping code does.
```

20. **Test coverage** - Fortran unit harnesses, the C++ harness (`test_wwdigi`, `test_contestlog`, `test_field`, `test_uilock`, `test_interlock`, `test_title`, `test_comment`, `test_migrate`,

`test_wav16`, `test_reply`, `test_ignore`, ...), and scripted GUI replays of both QSO roles.

28. **A new log file names the fork** (2026-09-04). The header line JTDX inherited from WSJT-X, `WSJT-X ADIF Export<eoh>`, written by `ADIF::addQSOToFile` when the log file is empty at the first QSO, now reads `JTDX_contest ADIF Export<eoh>` - the literal in both trees rather than `fork_name()`, because the log leaves the machine and must not carry the private build's name. Only a newly created `wsjtx_log.adi` gets it; an existing file keeps its header, and nothing in the program reads the line back.

Known open items (the project notes)

Deliberate, watched rather than fixed: `report()` grid capture not contest-gated inside `DecodedText` itself; `RCALL -> Tx3` can roger a grid remembered from an earlier cycle; AutoFilter stays off after leaving the waiting-caller behaviour; working the same station again on another band relies on message timestamps moving forward; and period timing / PTT-time hooks are untestable by replay - **on-air testing is the remaining step**.