

WSJT-X UDP message protocol — a field-tested reference

A practical, implementation-oriented summary of WSJT-X's UDP interface, as used by **wsjtx-mcp**. The authoritative source is `Network/NetworkMessage.hpp` in the WSJT-X repository; this document captures the wire details you actually need to encode and decode the messages, plus what was verified live.

Verified live on 2026-06-26 against WSJT-X reporting version **3.0.2** (Heartbeat header schema **2**, advertised max schema **3**). Real captured Status/Heartbeat datagrams are checked in as golden fixtures in `tests/test_golden.py`.

Transport

- **UDP**, default endpoint `127.0.0.1:2237` (WSJT-X → Settings → Reporting → UDP Server). May be unicast or a multicast group.
- WSJT-X is mostly a **broadcaster**: it pushes Status, Decode, QSOLogged, LoggedADIF, WSPRDecode, Clear, Close and a periodic **Heartbeat (every 15 s)**. There is no "get frequency / get mode" query — you learn state from Status.
- It honours a small inbound **control** set *only if* "**Accept UDP requests**" is enabled (off by default).
- Every control message must carry the target instance's **id**, which you only learn by first *receiving* a datagram from it. Control replies go to the **source address** the datagram arrived from.

Datagram framing

```
quint32 magic      = 0xADBCCBDA
quint32 schema      (1=Qt_5_0 broken, 2=Qt_5_2, 3=Qt_5_4)
quint32 message type
utf8      Id        (first payload field of every message)
...       type-specific payload
```

Everything is **big-endian**, serialized as Qt `QDataStream`. All floating-point fields are **doubles** (64-bit).

Type encodings (QDataStream, schema 3 / Qt_5_4)

Type	Wire format
quint8/32/64, qint32/64	big-endian fixed-width
bool	one byte 0x00/0x01
double	8-byte IEEE-754 (all floats are doubles)
utf8 (QByteArray)	quint32 length + bytes, no terminator. 0xFFFFFFFF = null , 0 = empty (distinct)
QTime	quint32 ms since midnight (0xFFFFFFFF = invalid)
QDateTime	qint64 Julian day + quint32 ms + quint8 timespec (0 local, 1 UTC, 2 offset → qint32); WSJT-X avoids time-zone forms

QColor	qint8 spec + 5xquint16 (alpha, r, g, b, pad); 8-bit channels scaled $c \ll 8$ c. Spec 0 (invalid) clears highlighting
--------	--

The quint32 fields **Frequency Tolerance**, **T/R Period**, and **Rx DF** use 0xFFFFFFFF as a "not applicable / no change" marker.

Messages

Direction is relative to a server: **Out** = WSJT-X → you; **In** = you → WSJT-X.

#	Message	Dir	Payload after 
0	Heartbeat	Out/In	max schema quint32, version utf8, revision utf8
1	Status	Out	dial freq quint64, mode, dx call, report, tx mode, tx enabled bool, transmitting bool, decoding bool, rx df quint32, tx df quint32, de call, de grid, dx grid, tx watchdog bool, sub-mode, fast mode bool, special-op quint8, freq tol quint32, T/R period quint32, config name, tx message
2	Decode	Out	new bool, time QTime, snr qint32, Δt double, Δf quint32, mode, message, low-confidence bool, off-air bool
3	Clear	Out/In	(In) window quint8: 0 band, 1 rx, 2 both
4	Reply	In	time QTime, snr qint32, Δt double, Δf quint32, mode, message, low-confidence bool, modifiers quint8
5	QSOLogged	Out	datetime-off QDateTime, dx call, dx grid, tx freq quint64, mode, report sent, report rcvd, tx power, comments, name, datetime-on QDateTime, op call, my call, my grid, exch sent, exch rcvd, ADIF prop mode
6	Close	Out/In	—
7	Replay	In	—
8	HaltTx	In	auto-tx-only bool

9	FreeText	In	text utf8, send bool
10	WSPRDecode	Out	new bool, time QTime, snr quint32, Δt double, freq quint64, drift quint32, callsign, grid, power quint32, off-air bool
11	Location	In	location utf8 (Maidenhead 4/6)
12	LoggedADIF	Out	ADIF text utf8
13	HighlightCallsign	In	callsign, bg QColor, fg QColor, highlight-last bool
14	SwitchConfiguration	In	configuration name utf8
15	Configure	In	mode, freq tol quint32, submode, fast mode bool, T/R period quint32, rx df quint32, dx call, dx grid, generate-messages bool

Enumerations

- **Special Operation Mode** (Status): 0 NONE, 1 NA VHF, 2 EU VHF, **3 FIELD DAY**, 4 RTTY RU, 5 WW DIGI, 6 FOX, 7 HOUND, 8 ARRL DIGI.
- **Reply.Modifiers** (bitmask): 0x00 none, 0x02 SHIFT, 0x04 CTRL/CMD, 0x08 ALT, 0x10 META, 0x20 KEYPAD, 0x40 group-switch.

Schema negotiation & compatibility

- Send a Heartbeat on startup advertising your max schema (3). The peer's Heartbeat carries *its* max schema; use `min(yours, theirs)` for outbound. A missing "max schema" field means assume schema 2.
- Two backward-compatibility rules: ignore **unknown message types** silently, and ignore **extra trailing fields** in known messages. wsjtx-mcp's parser also tolerates *missing* trailing fields (older builds) by returning a partial decode rather than erroring.

Hard limitations to design around

- **No dial-frequency control over UDP.** `Configure` sets mode/submode/Rx DF/T-R/freq-tolerance/DX call+grid, but **not** the dial frequency. QSY is a rig-control concern (Hamlib/CAT or the UI).
- **No "Enable Tx" over UDP** — and no UDP control of the **Auto Seq**, **Call 1st**, or **Hold Tx Freq** checkboxes either. You can `HaltTx`, but cannot press Enable Tx; transmission is *started by* `Reply` (answer a CQ) or `FreeText` with `Send=true`. The protocol is deliberately a "cooperative service, not a secondary user interface" (per the header).
- **Reply must exactly match a prior CQ/QRZ decode** — it is equivalent to double-clicking that line. Whether the *rest* of the QSO then runs automatically depends on WSJT-X's **"Auto Seq"** setting: with Auto Seq **on** (the usual FT8/FT4 default) WSJT-X sequences the full exchange to `QSOLogged`; with it **off**, `Reply` starts only the first transmission and the header's expectation is that the operator completes the QSO manually in the UI.
- **Strong for S&P, weak for RUN.** Because `Reply` only answers CQ/QRZ and there is no Enable-Tx toggle, *answering* CQs is fully automatable but *calling* CQ on a repeating cycle is not — that relies on WSJT-X's own Enable Tx, or a `FreeText` CQ re-sent each T/R period (without WSJT-X's standard sequencing).

- **One binder per UDP port** (a UDP/OS fact, not a WSJT-X limit). If JTAlert/GridTracker/N1MM own 2237, use a secondary UDP server, a multicast group, or a different host.