

The N3FJP TCP API — A Field-Tested Reference

A complete, developer-friendly guide, verified live against API version 2.2

Compiled for the contest-mcp project by Stefan Brunner (AE5VG)

2026-06-23

A community reference. This guide was written while building `contest-mcp` and is shared freely (the project is MIT-licensed) to give back to the amateur-radio and developer community. It reorganizes and completes the official N3FJP API page, and — importantly — records what we **verified live** against a running instance, including several places where today’s API (v2.2) differs from the public 0.9 documentation. Corrections and additions are welcome via issues / pull requests. This is an independent effort, not affiliated with or endorsed by Affirmatech / N3FJP.

Overview

Every program in the **N3FJP Software** suite — Amateur Contact Log (AC Log) and the 100-plus contest loggers — ships the **same** TCP Application Program Interface (API). Code against one and you have effectively coded against all of them. The API lets an external application behave exactly as a human operator would: type into the entry boxes, log QSOs, change band and mode, run dupe checks, search the log, and (optionally) receive a live push feed of everything happening in the program.

This document reorganizes and completes the official reference at <https://www.n3fjp.com/help/api.html> into something a developer can implement from end to end. Where the official page only *describes* a capability without showing the literal command, this document marks the tag as **(verify live)** so you know to confirm it against a running instance.

Enabling the API. In N3FJP, open **Settings → Application Program Interface**, confirm the form shows **version 0.8 or later**, and tick **TCP API Enabled**. Nothing else is required on the N3FJP side.

Verified live. The command and response details here were checked against *N3FJP’s ARRL Field Day Contest Log* v6.6.10, **API version 2.2** (2026). The API has evolved since the public 0.9 page: some commands were renamed or dropped, and an unknown command now returns `<CMD><CMD_NOT_FOUND></CMD>`. The companion machine-readable spec lists the exact, verified response tags and the differences from 0.9.

Transport and message format

The connection is plain **TCP**. **N3FJP is the server**; your application is the client. The default port is **1100**, though any port may be configured.

Don't confuse the API port with the networking port. N3FJP also has a multi-computer **Network** feature (Settings → Network) that listens on its own port (commonly 1000). That is a *different* protocol and does **not** answer API commands — connecting to it returns nothing. The API is the one under *Settings* → *Application Program Interface*. (Verified live: the API answered on 1100; the networking port on 1000 was silent to API commands.)

Each command you send must end with a **carriage-return + line-feed** (`\r\n`). To close the session cleanly, send a lone `\r\n` (or just close the socket).

Every command and every response is wrapped in a `<CMD>...</CMD>` envelope. The first token after `<CMD>` is the command (or response) identifier; any parameters follow as nested XML tags:

```
<CMD><COMMANDID><TAG>value</TAG><TAG2>value2</TAG2></CMD>
```

Three rules that matter in practice:

1. **Parse by scanning, not by line.** A single TCP packet can carry several `<CMD>...</CMD>` blocks back to back. Read continuously and split on `</CMD>`.
2. **Casing.** Command IDs are upper case. N3FJP's own parser upper-cases tags before matching, so tag case is not significant — but sticking to the documented casing keeps logs readable.
3. **Responses and notifications share the stream.** Once you opt into notifications (below), unsolicited event blocks arrive interleaved with the responses to your requests. A robust client routes each parsed block either to the request that is waiting for it or to a notification buffer.

A minimal, dependency-free client therefore needs only Python's standard-library `socket`, a background reader that accumulates bytes and emits complete blocks, and a tiny XML-ish tag extractor.

The text entry boxes

The entry boxes are the program's input fields. You can read them, write them, clear them, and move focus to them.

Action	Command
Read a box	<code><CMD><READ><CONTROL>TXTENTRYCALL</CONTROL></CMD></code>
Write a box	<code><CMD><UPDATE><CONTROL>TXTENTRYCALL</CONTROL><VALUE>W</VALUE></CMD></code>
Clear a box	<code><CMD><UPDATE><CONTROL>TXTENTRYCALL</CONTROL><VALUE></VALUE></CMD></code>
Focus a box	<code><CMD><SETFOCUS><CONTROL>TXTENTRYRSTR</CONTROL></CMD></code>

A read returns:

```
<CMD><READRESPONSE><CONTROL>TXTENTRYCALL</CONTROL><VALUE>W1AW</VALUE></CMD>
```

Two boxes you must not write. Never send a value to `TXTENTRYCOUNTRYWORKED`; let N3FJP derive the country (and CQ zone, ITU zone, continent, and prefix) from the call sign you place in the Call box. The spelling of country names must match N3FJP's database exactly, so deriving it is the only reliable path.

RST quirk. In the contest software, the RST boxes auto-fill with defaults only when `TXTENTRYRSTR` *receives focus*. They will therefore not be populated by the `CALLTAB` action alone. Either supply RST values yourself or set focus to `TXTENTRYRSTR`.

Action commands

Actions invoke the program’s own functions, exactly as if the operator pressed a key. Send them as:

<CMD><ACTION><VALUE>ENTER</VALUE></CMD>

Wait at least **5 milliseconds** after an action before sending the next command.

Action	Effect
ENTER	Logs the contact from the current box values, updates the lists, clears the form. The only action that returns a value — the number of records added (1 = logged, 0 = not).
CLEAR	Clears all entry boxes.
CALLTAB	Performs everything that tabbing out of the Call field does: dupe check (contest software), previous-contact listing and retrieval, and — in AC Log — call-book lookup, Watch List checks, and more. Send it immediately after writing the Call box.
DUPECHECK	Runs the dupe-check function and its associated events. Only meaningful in contest software, and redundant if you already fire CALLTAB.

ENTER responds:

<CMD><ENTERRESPONSE><VALUE>1</VALUE></CMD>

The in-memory QSO count updates immediately, but the on-disk log file may lag by 100 ms or more (a separate thread writes it). If you intend to read the file directly afterward, allow time for the write.

Field-tested gotcha (API 2.2). When N3FJP runs in **Network mode** (a multi-PC cluster sharing a master log), ENTERRESPONSE can come back with VALUE=0 **even though the QSO was logged** — the master-table commit is asynchronous, so the immediate count is zero. We confirmed the record *was* added (it appeared in SEARCH, QSOCOUNT incremented, and a repeat DUPECHECK reported it). **Don’t rely on ENTERRESPONSE’s value to decide success in networked mode** — instead compare QSOCOUNT before and after. Separately, if the networking server is unreachable, ENTER times out with a “*Server Failed to Respond*” dialog and logs nothing; run standalone or keep the networking node reachable.

The automatic-logging flow

This is the single most important sequence — the reason the API exists:

1. UPDATE TXTENTRYCALL with the call sign.

2. ACTION CALLTAB — fires dupe check and previous-contact lookups.
3. UPDATE the exchange boxes for the contest (see *Contest exchange fields*).
4. Make sure the program is on the **correct band and mode** (next section).
5. ACTION ENTER.
6. Read ENTERRESPONSE (and surface any DUPECHECKRESPONSE / CALLTABDUPEEVENT).

What CALLTAB gives you back (verified, API 2.2). A beat after you send ACTION CALLTAB, N3FJP returns a single, information-rich CALLTABEVENT — the whole call lookup in one block. For example, tabbing on AE5VG returned:

```
<CMD><CALLTABEVENT><CALL>AE5VG</CALL><BAND>20</BAND><MODE>DIG</MODE>
<MODETEST>DIG</MODETEST><COUNTRY>USA</COUNTRY><DXCC>291</DXCC><MYCALL>K6ABC</MYCALL>
<OPERATOR>AE5VG</OPERATOR><QSOCOUNT>0</QSOCOUNT><PFX>AE5</PFX><CONT>NA</CONT>
<CQZ>4</CQZ><ITUZ>7</ITUZ><LAT>31.9</LAT><LON>-96.3</LON><BEARING>89.0</BEARING>
<LONGPATH>269.0</LONGPATH><DISTANCE>1,199</DISTANCE></CMD>
```

If the call is a duplicate you also receive a CALLTABDUPEEVENT. This single event is usually all you need for “who am I working and have I worked them before” — no separate country/zone lookups required.

Band, mode, and frequency

Read the current band/mode/frequency:

```
<CMD><READBMF></CMD>
```

```
→ <CMD><READBMFRESPONSE><BAND>20</BAND><MODE>USB</MODE><MODETEST>PH</MODETEST><FREQ>14070000</FREQ></CMD>
```

MODETEST (mode-for-contest) is always CW, PH, or DIG and drives dupe checking. MODE is only populated when rig interface is enabled. From API 0.8 on, a READBMFRESPONSE is also **pushed automatically** whenever band, mode, or frequency changes.

You must set the correct band and mode before ENTER. How you do that depends on whether the operator has rig interface enabled:

- **Rig interface enabled** — change frequency (and thus band) with CHANGEFREQ:

```
<CMD><CHANGEFREQ><FREQ>14070000</FREQ></CMD>
```

A frequency in the CW or phone portion of the band makes the mode default to CW or PH. To prevent that, add <SUPPRESSMODEDEFAULT>TRUE</SUPPRESSMODEDEFAULT>. To force a non-default mode, send the mode command first, then CHANGEFREQ with SUPPRESSMODEDEFAULT set. N3FJP replies with CHANGEFREQRESPONSE.

- **Rig interface disabled** — you may write the band and mode boxes directly (zero-length values are ignored). This method does **not** work while rig interface is enabled, unless you first set <CMD><IGNORERIGPOLLS><VALUE>TRUE</VALUE></CMD> — and you **must** set it back to FALSE afterward.

Queries

Query	Command	Response
Program + version	<CMD><PROGRAM></CMD>	PROGRAMRESPONSE with program name and version
API version	<CMD><APIVERSION></CMD>	APIVERSIONRESPONSE with the version string
Next serial number	<CMD><NEXTSERIALNUMBER></CMD>	The next serial (serial-exchange contests only)
Open log file path	<CMD><FILEPATH></CMD>	<CMD><FILEPATHRESPONSE><VALUE>...\LogData
Settings file path	<CMD><SETTINGSPATH></CMD>	SETTINGSPATHRESPONSE
Shared folder path	<CMD><SETTINGSPATHSHARED></CMD>	SETTINGSPATHSHAREDRESPONSE
QSO rate + contest stats	<CMD><QSORATE></CMD>	QSORATERESPONSE with 20-/60-minute rates and labelled contest totals
Visible fields	<CMD><VISIBLEFIELDS></CMD>	the fields currently shown, with values
All fields	<CMD><ALLFIELDS></CMD>	every field, regardless of visibility, with values
User data	<CMD><READUSERDATA></CMD> (<i>verify live</i>)	CALL, NAME, STATE, SECTION, CLASS, COUNTRY, CQZ, ITUZ, LAT, LONG, CNTY, CONT

Sample QSORATERESPONSE (Field Day):

```
<CMD><QSORATERESPONSE><20MIN>3</20MIN><60MIN>1</60MIN><L1>Total CW Contacts</L1>
<V1>1</V1><L2>Total Phone Contacts</L2><V2>8</V2><L3>Total DIG Contacts</L3>
<V3>9</V3><L4>Total QSO Points</L4><V4>28</V4></CMD>
```

Dupe check (query only)

To ask whether a call is a dupe **without** triggering N3FJP's dupe actions:

```
<CMD><DUPECHECK><CALL>W1AW</CALL></CMD>
```

A duplicate returns details; a clean call returns an empty value:

```
<CMD><DUPECHECKRESPONSE><CALL>W1AW</CALL><VALUE>Duplicate! W1AW at 2/09 19:25 on 10 CW Rec# 4<
```

This is contest-software only. State QSO-party programs return nothing here — use the DUPECHECK *action*, or watch for the pushed CALLTABDUPEEVENT.

Listing and searching the log

For bulk retrieval, reading the Access database directly is faster, but the API can list and search too.

List the most recent QSOs:

Command	Returns
<code><CMD><LIST></CMD></code>	all records, primary fields only (Call, Date, Time On, Band, Mode, Mode-contest)
<code><CMD><LIST><VALUE>25</VALUE></CMD></code>	the 25 most recent, primary fields
<code><CMD><LIST><INCLUDEALL></CMD></code>	all records, every non-empty field
<code><CMD><LIST><INCLUDEALL><VALUE>20</VALUE></CMD></code>	the 20 most recent, every non-empty field

Search by any combination of criteria (omitted criteria are ignored):

```
<CMD><SEARCH><CALL>W1AW</CALL><MODE>CW</MODE></CMD>
```

```
<CMD><SEARCH><BAND>15</BAND><MODE>CW</MODE></CMD>
```

```
<CMD><SEARCH><DXCC>256</DXCC><CONFIRMED>TRUE</CONFIRMED></CMD>
```

- MODE is normalized to CW/PH/DIG.
- COUNTRYWORKED must match N3FJP's spelling exactly; prefer DXCC (the ADIF country number).
- CONFIRMED TRUE/FALSE filters confirmed/unconfirmed (mainly AC Log); omit for both.
- INCLUDEALL returns every non-empty field; MAXRECORDS caps the result — ideal for “have I worked this entity?” checks.

Entity status (API 0.7+) answers “is this DX new/worked/confirmed on this band and mode?” against the connected AC Log, returning one of:

Code	Meaning
ATNO	All-Time New One (not in log)
OW	worked on some band/mode
OC	confirmed on some band/mode
OWBMW	not confirmed anywhere, but worked this band+mode
OCBMW	confirmed elsewhere, worked (not confirmed) this band+mode
BMC	confirmed on this band and mode

Notifications (opt-in push feed)

By default the connection is request/response. Opt into push events to mirror the program's state live.

All field updates. After connecting, send:

```
<CMD><ALLUPDATES><VALUE>TRUE</VALUE></CMD>
```

Thereafter, every text-box change (whether the operator or your app caused it) is pushed back as a control-update block. Touching the VFO with rig interface on, for example, streams frequency/band/mode/zone/prefix updates. Send FALSE to stop.

Enter / Call-tab events (API 0.8+) are pushed automatically when the operator logs a contact or tabs from the Call field, including CALLTABDUPEEVENT when the tabbed call is a dupe.

DX spots, keydown events, and app-driven rig polling can also be relayed. The exact enabling tags for these are described but not shown literally in the official page; confirm them against a live instance (they are reachable through the escape hatch meanwhile).

Because MCP is request/response, **contest-mcp** enables notifications on demand and exposes a tool to **drain** the buffered event blocks rather than streaming them.

Adding records directly, and raw SQL

The log file is a Microsoft Access database; QSO records live in `tblContacts`. The **preferred** way to add a contact is always the ENTER action, because in the contest software it assigns points and multipliers on entry. Two lower-level paths exist for special cases:

Add direct via the API (field names use the `fld...` prefix):

```
<CMD><ADDDIRECT><EXCLUDEDUPES>TRUE</EXCLUDEDUPES><STAYOPEN>TRUE</STAYOPEN>
<fldCall>W1AW</fldCall><fldBand>20</fldBand><fldMode>SSB</fldMode>
<fldModeContest>PH</fldModeContest><fldDateStr>2026/06/27</fldDateStr>
<fldTimeOnStr>18:30</fldTimeOnStr><fldRstR>59</fldRstR><fldRstS>59</fldRstS>
<fldClass>2A</fldClass><fldSection>CT</fldSection></CMD>
```

Keep `STAYOPEN TRUE` while sending a batch (it holds the SQL connection open), then `FALSE` (or `<CMD><SQLCLOSE></CMD>`) on the last record. `EXCLUDEDUPES TRUE` skips a record only if Call, Date, Time-on (minute), Band, Mode, both grids, both counties, and State all match. Dates are YYYY/MM/DD, times HH:MM, with / and : separators — exactly.

Raw SQL against the database is possible via the API but must be treated as dangerous; an errant statement can delete or corrupt the whole log. After issuing SQL, you must `<CMD><SQLCLOSE></CMD>`. To make changes appear immediately:

- `<CMD><CHECKLOG></CMD>` — load only new records (fast; use after adds).
- `<CMD><OPENLOG></CMD>` — reload the whole log (use after edits).

contest-mcp policy. Direct adds and individual-record deletes are destructive and require an explicit **confirm**. Raw SQL that could wipe or overwrite the entire database is refused unless the operator turns on a dedicated, off-by-default configuration switch with a stern warning. See the project README.

Other capabilities

The API also exposes: a dialogue label you can write to (and optionally have spoken), text-to-speech, rig-offset read/enable/set, the “don’t send mode change with frequency” option, reading and setting the main form’s size/location, country-list lookup from a call sign, sending CW, and keying the rig (`RIGTX/RIGRX`, or CW com-port key down/up). CW and TX keying are out of scope for a logging assistant and, in **contest-mcp**, live behind the confirmed escape hatch only.

Contest exchange fields

contest-mcp’s log tool should make it easy to set the right exchange fields per contest. The digital-capable contests and their required fields:

Contest	Exchange fields
ARRL Field Day	Class / Section
Winter Field Day	Category (use Class) / Section
CQ WPX	RST / Serial received
CQ WW RTTY	RST / CQ Zone (auto from call) / State or Province
NCJ NAQP	Name / SPCNum
NCJ North American Sprint	Serial received / Name / SPCNum
ARRL RTTY Roundup	(see VISIBLEFIELDS)
ARRL School Club Roundup	RST / Class / SPCNum / Name (optional)
ARRL Kids Day	Name / Age / SPCNum
ARRL Rookie Roundup	Name / Check / SPCNum
Africa All Mode	RST / Serial received
Italian ARI Int'l DX	RST / SPCNum
Ten-Ten	Name / SPCNum / 10-10 number
VHF	Grid / RST (optional)
Worked All Europe	RST / Serial received
State QSO Parties	Varies: Name / State / County / Serial / Section / SPCNum

TXENTRYSPECNUM (State/Province/Country/Number) is filled in contest-specific ways; always confirm a contest's live field set with `<CMD><VISIBLEFIELDS></CMD>`.

Field-tested notes & gotchas (verified against API 2.2)

These are the things that cost us time and aren't obvious from the official 0.9 page. All were observed live against *N3FJP's ARRL Field Day Contest Log* v6.6.10 reporting **API version 2.2**.

The API version isn't 0.9 anymore. PROGRAM returns an APIVER tag — ours was 2.2. Read it from PROGRAM; there is **no** APIVERSION command (it returns CMD_NOT_FOUND).

Unknown commands answer back. An unrecognised command returns `<CMD><CMD_NOT_FOUND></CMD>` rather than silence. This is genuinely useful — you can probe whether a command exists, and tell “not supported” apart from “no response / wrong port”.

Two sockets, two purposes — don't mix them up. Port 1100 is the *API*. Port 1000 is N3FJP's multi-PC *networking*/“*server*” socket (the master-table cluster). Connecting to 1000 with API commands returns nothing. Any cluster node appears to answer API calls regardless of its client/server role, but you must point your client at the **API** port.

Commands renamed or gone since 0.9 (all return CMD_NOT_FOUND on 2.2):

0.9 / assumed	Use instead on 2.2
APIVERSION	the APIVER tag in PROGRAM
READQSOCOUNT	QSOCOUNT → <code><QSOCOUNTRESPONSE><VALUE>n</VALUE></code>
ALLFIELDSVALUES	ALLFIELDS already includes values

0.9 / assumed	Use instead on 2.2
READUSERDATA	not present; read individual fields, or CALLTABEVENT
CHECKENTITY / ENTITYSTATUS	entity-status tag could not be confirmed on this build — treat as unavailable

Response tags that differ from intuition:

- PROGRAM → <PGM>, <VER>, <APIVER> (not VERSION).
- QSORATE → adds <TOTSCORE> and <TOTQSOS> ahead of the documented fields.
- VISIBLEFIELDS / ALLFIELDS → **one response block per field**, each <...RESPONSE><CONTROL>...</CONTROL><...>
- READBMF → FREQ is a float string (e.g. 0.000000 with no rig connected).
- DUPECHECKRESPONSE's detail is a human-readable string that contains an embedded CRLF, e.g. Duplicate! AE5VG at 6/23 22:01 on 20 DIG\r\nRec# 1 — split envelopes on </CMD>, **not** on newlines, or you'll truncate it.

ENTER can report 0 yet still log in Network mode (see the gotcha box under *Action commands*). Verify with the QSOCOUNT delta, not the ENTERRESPONSE value.

CALLTAB is your friend. One CALLTABEVENT carries country, DXCC, both zones, prefix, continent, bearing, long-path, and distance (see *The automatic-logging flow*). It arrives a fraction of a second after the command, so give your read a ~1 second settle window.

Notifications are best-effort for app-driven changes. With ALLUPDATES TRUE, *operator-driven* UI changes are pushed, but on this build our own app-initiated UPDATES did **not** echo back as separate push events. Events caused by your own commands (like CALLTABEVENT) arrive **inline** in that command's reply, not in the async buffer.

Derived fields are read-only in practice. TXTENTRYCOUNTRYWORKED, TXTENTRYCQZONE, TXTENTRYITUZ, TXTENTRYIARUZONE, TXTENTRYCONTINENT, and TXTENTRYPREFIX are computed by N3FJP from the call — set the call and let it fill them.

Implementation checklist

1. Open a TCP socket to the configured host/port; do not assume loopback.
2. Send commands terminated with \r\n; never split a <CMD> across writes.
3. Read continuously; split the byte stream on </CMD>; parse each block's ID and nested tags.
4. Route blocks to waiting requests by response ID; buffer the rest as notifications.
5. For logging, follow the Call → CALLTAB → exchange → band/mode → ENTER flow and surface dupe responses.
6. Treat add-direct, deletes, raw SQL, and any TX/CW keying as destructive; guard whole-database operations behind an explicit configuration switch.

Credits, license, and contributing

This reference was compiled by **Stefan Brunner (AE5VG)** while building `contest-mcp`, with thanks to **Affirmatech / N3FJP** for the logging software and its open API, and to the authors

of community clients (notably `dslotter/wsjsx_to_n3fjp`) whose code helped confirm the wire format.

It is released under the project's **MIT license** — use it, quote it, and build on it freely. If you spot an error, find a command this guide marks uncertain, or test against a different N3FJP program/API version, please open an issue or pull request so the community reference stays accurate: <https://github.com/sbrunner-atx/contest-mcp/issues>.

The companion machine-readable spec gives the same information in a terse, structured form suitable for code generation.

Independent project; not affiliated with or endorsed by Affirmatech / N3FJP. “N3FJP” is the callsign and trademark of its owner.