

fldigi XML-RPC API (field-verified, 4.2.13)

fldigi XML-RPC API — A Field-Tested Reference

A clean, complete, **field-verified** reference to the XML-RPC control interface built into [fldigi](#). Every method below was enumerated **live** from a running instance via `fldigi.list`, so it reflects what the software actually exposes — not just what the wiki documents.

Tested against: `fldigi 4.2.13` (2026-09-09; identical to 4.2.11, 2026-06-23) — **174 methods** enumerated via `fldigi.list` / `fldigi.name_version` on 2026-06-23. Newer or older builds may add or remove methods — always call `fldigi.list` to confirm what *your* build supports. A terse, machine-readable catalog of all 174 is in [FLDIGI-API-SPEC.md](#).

Independent project — not affiliated with fldigi or the W1HKJ project. This reference is a free community resource maintained alongside [fldigi-mcp](#). `fldigi` is the work of Dave Freese W1HKJ and contributors. Corrections welcome — see the bottom.

1. Transport & connection model

- **Protocol:** XML-RPC over HTTP. `fldigi` is the **server**; your program is the client. Requests are HTTP POST to `/RPC2` (the `xmlrpc.client` default).
- **Endpoint:** `http://127.0.0.1:7362/` by default.
- **Command-line switches** that configure it:
 - `--xmlrpc-server-address HOSTNAME` — bind address (default loopback)
 - `--xmlrpc-server-port PORT` — port (default 7362)
 - `--xmlrpc-allow REGEX / --xmlrpc-deny REGEX` — allow/deny by method name
- **Always on** by default — no menu option is required to enable it.
- **Security:** the interface is unauthenticated and unencrypted. Keep it on loopback; for LAN/remote use bind a specific interface and tunnel over SSH, and use `--xmlrpc-allow/--xmlrpc-deny` to whitelist methods.

Minimal client (Python standard library — no third-party packages):

```
import xmlrpc.client
fldigi = xmlrpc.client.ServerProxy("http://127.0.0.1:7362/", allow_none=True)

print(fldigi.fldigi.name_version())      # 'fldigi 4.2.13'
print(fldigi.main.get_trx_status())      # 'rx' | 'tx' | 'tune'
fldigi.modem.set_by_name("BPSK31")
fldigi.main.set_frequency(14070000.0)    # Hz, as a float
```

The dotted method names map straight onto attribute access: `fldigi.main.get_frequency()` issues the XML-RPC call `main.get_frequency`.

2. Signature type codes

`fldigi.list` reports each method's signature as `return:args`. The codes:

Code	Type
n	nil / void (no value)
b	boolean
i	integer
d	double (float)
s	string
6	bytes (XML-RPC base64)
A	array
S	struct

Examples: `s:n` returns a string and takes no argument; `d:d` returns a double and takes a double; `6:ii` returns bytes and takes two ints; `n:s` returns nothing and takes a string.

3. Discovery

Method	Sig	Description
<code>fldigi.list</code>	<code>A:n</code>	Array of {name, signature, help} structs — every method this build exposes. Start here.
<code>fldigi.name_version</code>	<code>s:n</code>	Program name + version, e.g. <code>fldigi 4.2.13</code> .
<code>fldigi.version</code>	<code>s:n</code>	Version string only.
<code>fldigi.version_struct</code>	<code>S:n</code>	Version as a struct (major/minor/patch).
<code>fldigi.name</code>	<code>s:n</code>	Program name.
<code>fldigi.config_dir</code>	<code>s:n</code>	Path to the configuration directory.
<code>fldigi.terminate</code>	<code>n:i</code>	Quit fldigi. Bitmask: 1=save options, 2=log, 4=macros (see gotchas).

4. `modem.*` — operating mode (Op Mode) & modem settings

Selects the digital mode and tunes the modem's audio parameters. fldigi calls these “modems”; the on-screen menu is **Op Mode**.

Method	Sig	Description
<code>modem.get_name</code>	<code>s:n</code>	Current modem name, e.g. <code>BPSK31</code> .
<code>modem.get_names</code>	<code>A:n</code>	All available modem names.
<code>modem.get_id</code>	<code>i:n</code>	Current modem ID.
<code>modem.get_max_id</code>	<code>i:n</code>	Highest modem ID.
<code>modem.get_mode</code>	<code>s:n</code>	ADIF mode of the current modem (e.g. <code>PSK</code>).
<code>modem.get_submode</code>	<code>s:n</code>	ADIF submode (e.g. <code>PSK31</code>).

Method	Sig	Description
modem.get_io_names	A:n	Modems usable for KISS/ARQ I/O.
modem.set_by_name	s:s	Select modem by name; returns the previous name.
modem.set_by_id	i:i	Select modem by ID; returns the previous ID.
modem.get_carrier	i:n	Audio carrier (Hz).
modem.set_carrier	i:i	Set carrier; returns old.
modem.inc_carrier	i:i	Increment carrier; returns new.
modem.get_bandwidth	i:n	Modem bandwidth (Hz).
modem.set_bandwidth / modem.inc_bandwidth	i:i	Set / increment bandwidth.
modem.get_afc_search_range	i:n	AFC search range.
modem.set_afc_search_range / modem.inc_afc_search_range	i:i	Set / increment AFC range.
modem.get_quality	d:n	Signal quality [0:100].
modem.search_up / modem.search_down	n:n	Search for a signal up/down the waterfall.
modem.olivia.get_bandwidth / set_bandwidth	i:n / n:i	Olivia bandwidth.
modem.olivia.get_tones / set_tones	i:n / n:i	Olivia tones.

```
fldigi.modem.set_by_name("Olivia 8/500")
fldigi.modem.olivia.set_bandwidth(500)
fldigi.modem.olivia.set_tones(8)
print(fldigi.modem.get_quality())      # current copy quality
```

5. main.* — operating controls, frequency, T/R

The largest namespace: the on-screen control buttons, the dial frequency, and transmit/receive switching.

5.1 Status & frequency

Method	Sig	Description
main.get_status1	s:n	First status field (typically S/N).
main.get_status2	s:n	Second status field.
main.get_frequency	d:n	RF carrier frequency (Hz). Deprecated → rig.get_frequency .
main.set_frequency	d:d	Set RF carrier (Hz); returns old.
main.inc_frequency	d:d	Increment RF carrier; returns new.
main.get_wf_sideband	s:n	Waterfall sideband (USB/LSB).
main.set_wf_sideband	n:s	Set waterfall sideband.

5.2 Control buttons (get / set / toggle)

Each control below has three boolean forms — `get_ (b:n)`, `set_ (b:b, returns the old state)`, and `toggle_ (b:n, returns the new state)` — using the stem shown. For example AFC is `main.get_afc` / `main.set_afc` / `main.toggle_afc`.

- **AFC** — `main.*_afc` — automatic frequency control.
- **SQL** (squelch) — `main.*_squelch` — squelch on/off.
- **Rev** — `main.*_reverse` — reverse sideband.
- **Lock** — `main.*_lock` — transmit-frequency lock.
- **RxID** — `main.*_rsid` — RSID *reception* (auto-detect incoming mode).
- **TxID** — `main.*_txid` — RSID *transmission* (present in 4.2.x; absent from the older wiki).

Squelch **level** is separate and is a **double** (not a toggle): `main.get_squelch_level (d:n)`, `main.set_squelch_level (d:d, returns old)`, `main.inc_squelch_level (d:d, returns new)`. See the gotchas about sending a float, not an int.

5.3 Transmit / receive — keying

Method	Sig	Description
<code>main.get_trx_status</code>	<code>s:n</code>	rx / tx / tune.
<code>main.get_trx_state</code>	<code>s:n</code>	T/R state.
<code>main.tx</code>	<code>n:n</code>	Key the transmitter. 
<code>main.tune</code>	<code>n:n</code>	Transmit a steady tuning carrier. 
<code>main.rx</code>	<code>n:n</code>	Return to receive.
<code>main.abort</code>	<code>n:n</code>	Abort a transmit or tune.
<code>main.rx_only</code>	<code>n:n</code>	Force receive-only (disable Tx).
<code>main.rx_tx</code>	<code>n:n</code>	Restore normal Rx/Tx switching.
<code>main.run_macro</code>	<code>n:i</code>	Run macro by number — may transmit. 
<code>main.get_max_macro_id</code>	<code>i:n</code>	Highest macro number.

5.4 Timing helpers

Method	Sig	Description
<code>main.get_tx_timing</code>	<code>n:s</code>	Transmit duration for a test string (samples:rate:secs).
<code>main.get_char_rates</code>	<code>s:n</code>	Table of per-character rates.
<code>main.get_char_timing</code>	<code>n:i</code>	Transmit duration for a given character value.

5.5 Deprecated `main.*` (use the `rig.*` equivalents)

`main.get_sideband`, `main.set_sideband`, `main.rsid`, and the entire `main.`

`{get,set}_rig_{name,frequency,mode,modes,bandwidth,bandwidths}` family are flagged **[DEPRECATED]** in their own help text — use `main.get_wf_sideband` / the `rig.*` methods instead.

```
if fldigi.main.get_trx_status() == "rx":
    fldigi.text.add_tx("CQ CQ de AE5VG AE5VG K^r") # ^r = return to RX when done
    fldigi.main.tx()
```

6. rig.* — rig (CAT) control

Transceiver control via flrig / Hamlib / RigCAT / XML-RPC. Some methods only return meaningful values when rig interface is configured.

Method	Sig	Description
rig.get_frequency	d:n	RF carrier frequency (Hz). Preferred over main.get_frequency.
rig.set_frequency	d:d	Set frequency; returns old.
rig.get_mode	s:n	Current transceiver mode (e.g. USB).
rig.set_mode	n:s	Select a mode previously added via rig.set_modes.
rig.get_modes	A:n	Available rig modes.
rig.set_modes	n:A	Define the list of available modes (XML-RPC rig).
rig.get_bandwidth	s:n	Current bandwidth.
rig.set_bandwidth	n:s	Select a bandwidth previously added via rig.set_bandwidths.
rig.get_bandwidths	A:n	Available bandwidths.
rig.set_bandwidths	n:A	Define the list of available bandwidths.
rig.get_name / rig.set_name	s:n / n:s	XML-RPC rig name.
rig.get_notch / rig.set_notch	s:n / n:i	Notch filter (waterfall-driven).
rig.set_smeter / rig.set_pwrmeter	n:i	Push S-meter / power-meter values into fldigi.
rig.enable_qsy	n:i	Enable/disable (1/0) QSY for XML-RPC transceiver control.

7. log.* and logbook.* — QSO log / contest fields

log.* reads/writes the fields on fldigi's main logging panel; logbook.* returns ADIF from the open logbook.

Gettable fields (s:n): frequency, time_on, time_off, date_on, date_off, call, name, rst_in, rst_out, serial_number, serial_number_sent, exchange, state, province, country, qth, band, notes, locator, az (and get_sideband, deprecated).

Settable fields (n:s): call, name, qth, locator, rst_in, rst_out, serial_number, exchange, set_contest_counter.

Method	Sig	Description
log.get_<field>	s:n	Read a field (see list above).
log.set_<field>	n:s	Write a settable field.

Method	Sig	Description
log.set_contest_counter	n:s	Set the starting contest serial.
log.clear	n:n	Clear all log fields.
logbook.last_record	s:n	ADIF of the most recent logbook record.
logbook.all_records	s:n	ADIF of the entire open logbook.

8. The data plane — text.*, rx.*, tx.*, rxtx.*

This is what makes full automation possible: **decoded received text** and **outgoing text to encode**.

Method	Sig	Description
text.get_rx_length	i:n	Number of characters in the RX widget.
text.get_rx	6:ii	Range (start, length) of decoded RX text — bytes (see gotchas).
text.clear_rx	n:n	Clear the RX widget.
text.add_tx	n:s	Append text to the TX widget (does not transmit).
text.add_tx_bytes	n:6	Append a byte string to the TX widget.
text.add_tx_queue	n:s	Append to the TX transmit queue (fldigi spells it “queue”).
text.clear_tx	n:n	Clear the TX widget.
rx.get_data	6:n	All new RX data since the last call (bytes) — ideal for polling.
tx.get_data	6:n	All new TX data since the last call (bytes).
rxtx.get_data	6:n	Combined RX+TX stream since the last call (bytes).

```
n = fldigi.text.get_rx_length()
chunk = fldigi.text.get_rx(0, n)
print(chunk.data.decode("utf-8", "replace"))
```

xmlrpc.client.Binary
decode bytes -> text

9. spot.* — PSK Reporter / autospotter

Method	Sig	Description
spot.get_auto / set_auto / toggle_auto	b:n / b:b / b:n	Autospotter on/off.
spot.pskrep.get_count	i:n	Callsigns spotted this session.

10. wefax.* — weather fax

All return a string (empty on success/timeout, otherwise an error/status).

Method	Sig	Description
wefax.state_string	s:n	Engine state (tx + rx).
wefax.skip_apt / wefax.skip_phasing	s:n	Skip APT / phasing during reception.
wefax.start_manual_reception	s:n	Start manual-mode reception.
wefax.end_reception	s:n	End reception.
wefax.set_max_lines	s:i	Max lines for a received image.
wefax.set_adif_log	s:b	Log sent/received images to ADIF.
wefax.get_received_file	s:i	Wait (with timeout) for the next received file; name or empty.
wefax.send_file	s:si	Transmit an image file. △ Empty on OK, else error.
wefax.set_tx_abort_flag	s:n	Cancel an image transmission.

11. navtex.* — NAVTEX / SITOR-B

Method	Sig	Description
navtex.get_message	s:i	Next message within a timeout (seconds); empty on timeout.
navtex.send_message	s:s	Transmit a NAVTEX/SITOR-B message. △ Empty on OK, else error.

12. NBEMS plumbing — flmsg.* and io.*

For flmsg integration and switching the ARQ/KISS I/O port. Niche; most operators won't need these.

Method	Sig	Description
flmsg.online / flmsg.available / flmsg.transfer	n:n	flmsg presence / data-available / transfer signals.
flmsg.squelch	b:n	Squelch state (flmsg view).
flmsg.get_data	6:n	All RX data since last query (bytes).
main.flmsg_online / _available / _transfer / _squelch	—	main.* aliases of the above.
io.in_use	s:n	Which I/O port is active (ARQ/KISS).
io.enable_kiss / io.enable_arq	n:n	Switch the I/O port.

13. Transmit safety ⚠

A handful of methods put RF on the air. Any automation must treat these with care — **a licensed control operator is responsible for every transmission.**

Keying methods (will transmit):

- main.tx — key the transmitter

- `main.tune` — steady tuning carrier
- `main.run_macro` — a macro may contain a transmit
- `wefax.send_file` — transmit a fax image
- `navtex.send_message` — transmit a NAVTEX message

Always safe (take the station off the air): `main.rx`, `main.abort`, `main.rx_only`.

Practical guidance: - **fldigi does not auto-stop after the TX buffer drains** — it stays keyed. Either append the ^r control to your TX text (returns to RX when sending finishes) or call `main.rx` afterward. - Default your automation to receive; gate keying behind explicit operator consent (this is exactly what `fldigi-mcp` does with its `callsign gate` and `per-transmit approval`). - Use `--xmlrpc-deny` to hard-block keying methods in shared/unattended setups.

14. Field-tested notes & gotchas

Things that bit us (or surprised us) verifying against fldigi 4.2.13:

1. **Binary returns are base64, not plain strings.** Anything with signature 6: (e.g. `text.get_rx`, `rx.get_data`) comes back as `xmlrpc.client.Binary` — read `.data` and `.decode()` it. Treat RX/TX data as bytes; decoded text can contain non-ASCII.
2. **Doubles vs ints matter.** `main.set_frequency` and `main.set_squelch_level` are d:d — send a **float**. Passing an int can yield a type error from fldigi. Likewise `carrier/bandwidth` are i:i (ints). Coerce to the exact type.
3. **allow_none=True.** Many methods return `nil (n:)`; construct your `ServerProxy` with `allow_none=True` to avoid marshalling errors.
4. **Setters return the old value.** `set_*` typically returns the previous value, and `toggle_*` returns the *new* one — don't mistake the old value for a failure.
5. **This build exposes more than the public wiki.** Present in 4.2.11 and 4.2.13 but absent from the older [wiki](#): `main.{get,set,toggle}_txid (TxID)`, `main.rx_only / main.rx_tx`, `rig.get_frequency`, `rig.enable_qsy`, `rig.set_smeter / rig.set_pwrmeter`, `modem.get_mode / get_submode / get_io_names`, `log.get_date_on/off`, `log.set_rst_in/out`, `log.set_contest_counter`, `logbook.last_record / all_records`, `main.get_tx_timing / get_char_rates / get_char_timing`, `text.add_tx_queue / add_tx_bytes`, and the `flmsg.* / io.*` families. **Always fldigi.list to confirm your build.**
6. **Deprecated methods are flagged in their own help text.** `main.get_frequency` → `rig.get_frequency`; `main.get_sideband` → `main.get_wf_sideband`; `main.rsid` → `main.{get,set,toggle}_rsid`; the `main.*_rig_*` family → `rig.*`. They still work but may be removed.
7. **Modem name ≠ ADIF mode.** `modem.get_name` returns fldigi's modem label (BPSK31); `modem.get_mode / get_submode` return the ADIF mode/submode (PSK / PSK31). Log with the ADIF values.
8. **Frequency is the dial, not the audio carrier.** `main/rig` frequency is the RF carrier in Hz; the *audio* offset is `modem.get_carrier` (Hz). The on-air frequency ≈ dial + carrier (per sideband).
9. **rig.* reads depend on rig interface.** Without CAT/flrig configured, `rig.get_mode/get_bandwidth` may be empty and `set_frequency` only updates fldigi's display, not a radio.
10. **fldigi.terminate bitmask.** The help says "0=options; 1=log; 2=macros"; in practice these are **bit values** — pass 1 to save options, 2 log, 4 macros, OR them together (e.g. 3 = options+log).
11. **Array parameters are one array.** `rig.set_modes`, `rig.set_bandwidths` and their deprecated `main.set_rig_*` twins (signature n:A) take a single XML-RPC array; spreading the list into positional strings returns type error. Verified 4.2.13.
12. **The two timing calls lie about their types.** `main.get_tx_timing` is listed as n:s and `main.get_char_timing` as n:i, but the live build rejects a string and an int and accepts a **base64**

- parameter, returning a string samples : sample rate : seconds. `main.get_char_rates` can take several seconds to answer. Verified 4.2.13.
13. `rig.take_control` / `rig.release_control` **do not exist** in any 4.2.x build (they appear in older third-party write-ups). Use `rig.enable_qsy`.
 14. `fldigi.list` **repeats two rows** (`log.set_rst_in`, `log.set_rst_out`), so it reports 176 entries for 174 methods.
 15. **4.2.13 is API-identical to 4.2.11**. Names, signatures and help text match row for row (diffed 2026-09-09). The behavioural notes above hold on both.

15. Worked example — send, then read the reply

```
import time, xmlrpc.client
f = xmlrpc.client.ServerProxy("http://127.0.0.1:7362/", allow_none=True)

# 1. set up
f.modem.set_by_name("BPSK31")
f.main.set_frequency(14070000.0)           # float!

# 2. transmit (^r returns to RX when the buffer drains)
f.text.clear_tx()
f.text.add_tx("CQ CQ de AE5VG AE5VG K^r")
f.main.tx()

# 3. wait for the reply, then read newly-decoded text
time.sleep(8)
data = f.rx.get_data()                    # bytes since last call
print(data.data.decode("utf-8", "replace"))
```

Credits, license & contributing

Maintained as a free community resource by **Stefan Brunner (AE5VG)** alongside [fldigi-mcp](#). Released under the **MIT License** — use it, fork it, quote it.

This is an **independent project and is not affiliated with, nor endorsed by, the fldigi / W1HKJ project**. fldigi is © Dave Freese W1HKJ and contributors.

Found an error, a missing method, or different behavior on your build? Please open an **issue or pull request** at <https://github.com/sbrunner-atx/fldigi-mcp/issues> — include your fldigi version (`fldigi.name_version`) and, ideally, the relevant `fldigi.list` entry. 73!