

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
<code>n</code>	nil / void (no value)
<code>b</code>	boolean
<code>i</code>	integer
<code>d</code>	double (float)
<code>s</code>	string
<code>6</code>	bytes (XML-RPC base64)
<code>A</code>	array
<code>S</code>	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 <code>{name, signature, help}</code> 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: <code>1</code> =save options, <code>2</code> =log, <code>4</code> =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.

Method	Sig	Description
<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. PSK).
<code>modem.get_submode</code>	<code>s:n</code>	ADIF submode (e.g. PSK31).
<code>modem.get_io_names</code>	<code>A:n</code>	Modems usable for KISS/ARQ I/O.
<code>modem.set_by_name</code>	<code>s:s</code>	Select modem by name; returns the previous name.
<code>modem.set_by_id</code>	<code>i:i</code>	Select modem by ID; returns the previous ID.
<code>modem.get_carrier</code>	<code>i:n</code>	Audio carrier (Hz).
<code>modem.set_carrier</code>	<code>i:i</code>	Set carrier; returns old.
<code>modem.inc_carrier</code>	<code>i:i</code>	Increment carrier; returns new.
<code>modem.get_bandwidth</code>	<code>i:n</code>	Modem bandwidth (Hz).
<code>modem.set_bandwidth</code> / <code>modem.inc_bandwidth</code>	<code>i:i</code>	Set / increment bandwidth.
<code>modem.get_afc_search_range</code>	<code>i:n</code>	AFC search range.
<code>modem.set_afc_search_range</code> / <code>modem.inc_afc_search_range</code>	<code>i:i</code>	Set / increment AFC range.
<code>modem.get_quality</code>	<code>d:n</code>	Signal quality [0:100] .
<code>modem.search_up</code> / <code>modem.search_down</code>	<code>n:n</code>	Search for a signal up/down the waterfall.
<code>modem.olivia.get_bandwidth</code> / <code>set_bandwidth</code>	<code>i:n</code> / <code>n:i</code>	Olivia bandwidth.
<code>modem.olivia.get_tones</code> / <code>set_tones</code>	<code>i:n</code> / <code>n:i</code>	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
<code>main.get_status1</code>	<code>s:n</code>	First status field (typically S/N).
<code>main.get_status2</code>	<code>s:n</code>	Second status field.

Method	Sig	Description
<code>main.get_frequency</code>	<code>d:n</code>	RF carrier frequency (Hz). Deprecated → <code>rig.get_frequency</code> .
<code>main.set_frequency</code>	<code>d:d</code>	Set RF carrier (Hz); returns old.
<code>main.inc_frequency</code>	<code>d:d</code>	Increment RF carrier; returns new.
<code>main.get_wf_sideband</code>	<code>s:n</code>	Waterfall sideband (<code>USB</code> / <code>LSB</code>).
<code>main.set_wf_sideband</code>	<code>n:s</code>	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** (squellch) — `main.*_sqlch` — squellch 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).

Squellch **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>	<code>rx</code> / <code>tx</code> / <code>tune</code> .
<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 (<code>samples:rate:secs</code>).
<code>main.get_char_rates</code>	<code>s:n</code>	Table of per-character rates.

Method	Sig	Description
<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
<code>rig.get_frequency</code>	<code>d:n</code>	RF carrier frequency (Hz). Preferred over <code>main.get_frequency</code> .
<code>rig.set_frequency</code>	<code>d:d</code>	Set frequency; returns old.
<code>rig.get_mode</code>	<code>s:n</code>	Current transceiver mode (e.g. <code>USB</code>).
<code>rig.set_mode</code>	<code>n:s</code>	Select a mode previously added via <code>rig.set_modes</code> .
<code>rig.get_modes</code>	<code>A:n</code>	Available rig modes.
<code>rig.set_modes</code>	<code>n:A</code>	Define the list of available modes (XML-RPC rig).
<code>rig.get_bandwidth</code>	<code>s:n</code>	Current bandwidth.
<code>rig.set_bandwidth</code>	<code>n:s</code>	Select a bandwidth previously added via <code>rig.set_bandwidths</code> .
<code>rig.get_bandwidths</code>	<code>A:n</code>	Available bandwidths.
<code>rig.set_bandwidths</code>	<code>n:A</code>	Define the list of available bandwidths.
<code>rig.get_name</code> / <code>rig.set_name</code>	<code>s:n</code> / <code>n:s</code>	XML-RPC rig name.
<code>rig.get_notch</code> / <code>rig.set_notch</code>	<code>s:n</code> / <code>n:i</code>	Notch filter (waterfall-driven).
<code>rig.set_smeter</code> / <code>rig.set_pwrmeter</code>	<code>n:i</code>	Push S-meter / power-meter values into fldigi.
<code>rig.enable_qsy</code>	<code>n:i</code>	Enable/disable (<code>1/0</code>) QSRY 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
<code>log.get_<field></code>	<code>s:n</code>	Read a field (see list above).
<code>log.set_<field></code>	<code>n:s</code>	Write a settable field.
<code>log.set_contest_counter</code>	<code>n:s</code>	Set the starting contest serial.
<code>log.clear</code>	<code>n:n</code>	Clear all log fields.
<code>logbook.last_record</code>	<code>s:n</code>	ADIF of the most recent logbook record.
<code>logbook.all_records</code>	<code>s:n</code>	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
<code>text.get_rx_length</code>	<code>i:n</code>	Number of characters in the RX widget.
<code>text.get_rx</code>	<code>6:ii</code>	Range (<code>start</code> , <code>length</code>) of decoded RX text — bytes (see gotchas).
<code>text.clear_rx</code>	<code>n:n</code>	Clear the RX widget.
<code>text.add_tx</code>	<code>n:s</code>	Append text to the TX widget (does not transmit).
<code>text.add_tx_bytes</code>	<code>n:6</code>	Append a byte string to the TX widget.
<code>text.add_tx_queue</code>	<code>n:s</code>	Append to the TX transmit queue (fldigi spells it “queue”).
<code>text.clear_tx</code>	<code>n:n</code>	Clear the TX widget.
<code>rx.get_data</code>	<code>6:n</code>	All new RX data since the last call (bytes) — ideal for polling.
<code>tx.get_data</code>	<code>6:n</code>	All new TX data since the last call (bytes).
<code>rxtx.get_data</code>	<code>6:n</code>	Combined RX+TX stream since the last call (bytes).

```
n = fldigi.text.get_rx_length()
chunk = fldigi.text.get_rx(0, n)           # xmlrpc.client.Binary
print(chunk.data.decode("utf-8", "replace")) # decode bytes -> text
```

9. `spot.*` — PSK Reporter / autospotter

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

10. `wefax.*` — weather fax

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

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

11. `navtex.*` — NAVTEX / SITOR-B

Method	Sig	Description
<code>navtex.get_message</code>	<code>s:i</code>	Next message within a timeout (seconds); empty on timeout.
<code>navtex.send_message</code>	<code>s:s</code>	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
<code>flmsg.online</code> / <code>flmsg.available</code> / <code>flmsg.transfer</code>	<code>n:n</code>	flmsg presence / data-available / transfer signals.
<code>flmsg.squelch</code>	<code>b:n</code>	Squelch state (flmsg view).
<code>flmsg.get_data</code>	<code>6:n</code>	All RX data since last query (bytes).
<code>main.flmsg_online</code> / <code>_available</code> / <code>_transfer</code> / <code>_squelch</code>	—	<code>main.*</code> aliases of the above.
<code>io.in_use</code>	<code>s:n</code>	Which I/O port is active (ARQ/KISS).
<code>io.enable_kiss</code> / <code>io.enable_arq</code>	<code>n:n</code>	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_pwr_meter`, `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"))
```

16. PROPOSED methods — `browser.*` and `rsid.*` (patches offered upstream, not merged)

Status: proposed, in no released fldigi. Two patches, both offered to Dave W1HKJ on w1hkj/fldigi issue 55: the Signal Browser one on 11 September 2026, the RSID one on 16 September. Names, signatures and struct fields may change before or when they are merged; this section is updated when they land or are reworked. Both ship in [patches/](#) in fldigi-mcp and apply to fldigi's SourceForge git HEAD; the RSID patch applies on top of the browser patch.

Detect, do not assume. These methods appear in `fldigi.list` only on a build that carries the patch. Check that list before calling them; the `browser` and `rsid` tools do, and answer with a hint instead of an error on a stock build.

Why the browser. fldigi's Signal Browser (the multi-channel PSK/RTTY/CW decoder bank behind the left-hand panel and View › Signal browser) copies stations far too weak for a spectrum to rank, and has no XML-RPC surface in 4.2.13. [patches/fldigi-4.2.13-browser-xmlrpc.patch](#):

Method	Signature	Notes
<code>browser.get_channels</code>	<code>A:n</code>	Array of structs <code>{channel:int, freq:int, active:bool, text:string}</code> : one per channel that has printed since the last clear. <code>freq</code> is the audio frequency in Hz; <code>active</code> says the channel currently holds a signal; <code>text</code> is everything decoded on the channel since <code>browser.clear</code> , not trimmed to the widget width (capped at 8192 chars, oldest dropped), with the decoded line breaks kept (the PSK viewer used to turn them into spaces before the widget saw them) and a newline where the channel lost and regained a signal. Empty when the current modem

Method	Signature	Notes
		has no browser (Olivia, MFSK, ...) or nothing has printed.
<code>browser.clear</code>	<code>n:n</code>	Clears every channel on screen and in the buffer above.

Why RSID. Naming a mode from the spectrum is guesswork for the hopping modes, and fldigi already knows the answer whenever the other station sends RSID. In notify-only mode (RxID on, “do not change modem”) the detector reports without switching the modem, but only to a dialog and the status line. [patches/fldigi-4.2.13-rsid-hits.patch](#):

Method	Signature	Notes
<code>rsid.get_hits</code>	<code>A:n</code>	Array of structs <code>{utc:int, mode:string, hz:double}</code> : every RSID burst the detector accepted since <code>rsid.clear</code> , with the modem name as <code>modem.set_by_name</code> takes it and the audio frequency. Recorded whether or not RSID is in notify-only mode, so a program can leave the modem alone and still learn what was announced. Oldest entries drop after 1000.
<code>rsid.clear</code>	<code>n:n</code>	Forgets the recorded bursts.

Gotcha 16: the channel number is a slot in the decoder bank, not a frequency; under the browser’s ascending-order display the on-screen row and the slot differ, which is why the patch keeps its own per-slot frequency. Read `freq`, not `channel`.

Gotcha 17: `rsid.get_hits` records what the detector accepted, which is filtered by fldigi’s own RSID receive-mode exclusions; a mode you told fldigi to ignore never appears. End-of-transmission bursts are not recorded. With notify-only off, a hit is also the moment the modem changed, so a program reading the list should re-read `modem.get_name` rather than assume it still owns the modem.

Credits, license & contributing

Maintained as a free community resource by **Stefan Brunner (AE5VG)** alongside [fldigi-mcp](#). Released under the **GPL-3.0-or-later** — 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!