

PhantomSDR-Plus

Vollständige Dokumentation

16 Dokumente · erstellt 2026-10-03

Inhalt

1. README	8
Wichtigste Funktionen	8
Unterstützte Hardware	10
Schnellstart	11
Leistungsmessungen	12
Verwendung	12
Konfiguration	13
Wasserfall-Untergrenze — waterfall.sh	14
Anpassung	15
Dokumentation	17
Online-WebSDR-Liste	17
Fehlerbehebung	17
Mitwirken	18
Lizenz	18
Autoren und Danksagungen	18
Links	18
Unterstützung	18
Leistungstipps	18
2. Installationsanleitung	19
Inhaltsverzeichnis	19
Systemanforderungen	19
Vorbereitung vor der Installation	20
Abhängigkeiten installieren	21
Node.js und npm installieren	22
OpenCL installieren (optional, aber empfohlen)	23
PhantomSDR-Plus installieren	25
Opus-Audiocodec installieren	30
Autorun-Spot-Reporter (FT8/FT4/WSPR)	31
Emulation von KiwiSDR-Clients (optional)	33
Konfiguration	34
Gerätespezifische SDR-Einrichtung	37
Test und Überprüfung	40
Automatischen Start einrichten	41
CPU-Überhitzungsschutz	43
Fehlerbehebung	45
Leistungsoptimierung	49
PhantomSDR-Plus aktualisieren	49
Sichern und Wiederherstellen	52
Sicherheitsüberlegungen	53
Hilfe erhalten	53
Anhang A: vollständige Abhängigkeitsliste	54
Anhang B: Konfigurationsbeispiele	55
3. Admin-Panel einrichten	57
Überblick	57
Was das Admin-Panel bietet	57

Voraussetzungen	58
Schritt 1 — Dateien	58
Schritt 2 — Das Einrichtungsskript ausführen	58
Schritt 3 — Starten / Stoppen / Neu starten	59
Schritt 4 — Firewall-Port öffnen	61
Schritt 5 — Kick-Funktion	62
Konfiguration — admin_config.json	62
Die Einstellung sdr_host	63
proxy.py — was es ist und wann Sie es brauchen	63
Ports nach der Einrichtung ändern	64
Nützliche manuelle Befehle	64
Protokolldateien	65
Zugriffsübersicht	67
Löschen von Chatnachrichten	67
Wasserfall-Rundnachrichten	68
Systemgraphen	68
Thermal Guard	70
Spot Reporting (Aurorun FT8/FT4/J58/WSPR)	72
Fehlerbehebung	78
4. Thermal Guard	81
Inhalt	81
1. Was er tut	81
2. Schnellstart	82
3. Wie er entscheidet	82
4. Die vier Modi	83
5. Den Modus wählen	85
6. Konfigurationsreferenz	86
7. Betrieb ohne Admin-Panel	86
8. Die Throttle-Stufe ohne root aktivieren	87
9. Unbeaufsichtigter Betrieb	88
10. Testen, bevor Sie ihm vertrauen	89
11. Das Protokoll lesen	90
12. Lockout und Ratenbegrenzung	90
13. Fehlersuche	91
14. Abschalten oder entfernen	91
Siehe auch	92
5. Verbindungslimits	93
Inhalt	93
1. Wogegen Sie sich tatsächlich verteidigen	93
2. Die drei Schichten	94
3. Limits pro Adresse	94
4. Untätige Verbindungen	96
5. Der Kernel-Schutz	96
6. Konfigurationsreferenz	97
7. Was ein abgewiesener Hörer sieht	98
8. Logs und Zähler lesen	99
9. Ihre Zahlen wählen	99
10. Fallen, die man kennen sollte	100
11. Alles wieder abschalten	101

6. Mehrere Empfänger	102
Inhalt	102
1. Wie es funktioniert	102
2. Zwei Arten, die Empfänger zu veröffentlichen	103
3. Einen zweiten Empfänger hinzufügen	103
4. receivers.toml	105
5. Starten und Stoppen	106
6. Was die Hörer sehen	106
7. Das S-Meter oberhalb von 30 MHz	107
8. Sicherheit	107
9. Wie viele Empfänger passen	108
10. Fallen, die man kennen sollte	108
7. Statistikserver	109
Übersicht	109
Voraussetzungen	109
Installationsmethoden	109
Konfigurieren Sie Ihre PhantomSDR-Anwendung	112
Die Installation testen	114
Dienstverwaltung	115
Manuelle Installation (Option 2)	115
Den Port nach der Installation ändern	117
Fehlerbehebung	117
Was Sie sehen werden	120
Temperaturerkennung	120
Funktionsübersicht	121
Sicherheitshinweise	121
Support & Ressourcen	121
8. Decoder	124
Inhaltsverzeichnis	124
1. Einen Decoder starten	124
2. FT8	127
3. FT4-FT2	128
4. JS8	129
5. CW — Morsetelegrafie	131
6. QRSS Grabber	132
7. WSPR	135
8. HF-FAX / WEFAX	136
9. NAVTEX	138
10. FSK / RTTY — einschließlich PSK31 und Olivia	139
11. SSTV	142
12. Allgemeine Tipps	144
9. RADE README	146
Installation — der kurze Weg	146
Inhaltsverzeichnis	146
Funktionsweise	147
Architekturüberblick	148
RADEL gegenüber RADEU	148
Voraussetzungen	149

Installation von Hand	149
Die gepatchten Dateien ausbringen	149
Steuerung des Sidecars	150
Port 8074	151
RADE im Browser nutzen	151
Überprüfung und Fehlersuche	152
Umgebungsvariablen	153
Geänderte Dateien	154
Zusammenfassung des Signalflusses	154
Fehlerbehebung	155
Gleichzeitigkeit auf eigener Hardware messen	157
RADE v1 aktualisieren	159
10. RADE Allgemeine Installation (Linux)	162
Schritt 1 — Systempakete	162
Schritt 2 — Python-Pakete	163
Schritt 3 — Das radae-Repository klonen und bauen	164
Schritt 4 — Die Dekodierkette überprüfen	165
Schritt 5 — Das PhantomSDR-Plus-Frontend bauen	166
Schritt 6 — Steuerung des Sidecars	166
rade.sh wird weiterhin für den eigenständigen Betrieb mitgeliefert — das Sidecar ohne den PhantomSDR-Plus-Launcher zu betreiben oder es allein zu testen. Seine Befehle, sein Protokoll und die zugehörigen Watchdog-Fallstricke stehen unter Steuerung des Sidecars.	166
Schritt 7 — Port 8074 öffnen	166
Schritt 8 — Den Server starten	167
Schritt 9 — RADE starten	168
Schritt 10 — RADE im Browser nutzen	168
Hinweis zur CPU-Last	168
RADE künftig aktualisieren	169
11. Varianten bearbeiten	170
1. Was die Varianten sind	170
2. Welche Datei was macht	171
3. Eine Änderung, die nur einige Varianten betrifft	173
4. Wo die Variantenweichen sitzen	173
5. Weitere wissenswerte Stellschrauben	174
6. Nach dem Bearbeiten — neu bauen	175
7. Wie Sie Ihre Änderung prüfen, bevor Sie ihr vertrauen	175
12. Benutzerhandbuch	177
Inhaltsverzeichnis	177
Einführung	177
Erste Schritte	178
Überblick über die Oberfläche	179
Grundlegende Bedienung	180
Erweiterte Funktionen	183
Demodulationsarten	187
Digitale Decoder	190
Tastenkürzel	193
Lesezeichen	194
Nutzung auf Mobilgeräten	195
Tipps und bewährte Praktiken	196

Fehlerbehebung	197
Häufig gestellte Fragen	198
Ressourcen	199
Anhang: übliche Frequenzen	200
Glossar	201
13. KiwiSDR-Client-Emulation	202
Inhaltsverzeichnis	202
1. Was ein Kiwi-Client bekommt	202
2. Die Brücke installieren	203
3. Einschalten	203
4. Konfigurationsreferenz	204
5. Einen Client verbinden	204
6. Lautstärke	204
7. Das S-Meter	205
8. Wasserfall und Spektrum	206
9. Wenn etwas nicht stimmt	207
10. Wie weit das geprüft wurde	208
14. Empfangsdiversität	209
Was es tut und was nicht	209
Schnelleinstieg	210
Einen WebSDR verwenden	211
Das Feld lesen	212
Remote SNR trim	214
Die Wahl des zweiten Empfängers	214
Fehlersuche	215
Grenzen	215
15. Transceiver-Steuerung	217
Was Sie brauchen	217
Unterstützte Empfänger	217
Was es tut und was nicht	218
Schnellstart	218
Den Verbindungsweg wählen	219
Serielle Einstellungen	220
Synchronisierung	220
Betriebsarten	221
Filterbreite	222
Stummschaltung beim Senden	222
Frequenzversatz	223
Das Menü Rig	223
Linux	223
Windows	224
TCI-CAT auf der Empfängerseite	224
Für Empfängerbetreiber	234
Für Entwickler: die Seitenschnittstelle	234
Fehlerbehebung	235
Bekannte Einschränkungen	235
16. Projektstruktur	237
Inhaltsverzeichnis	237

Verzeichnisbaum	238
Wurzelverzeichnis	251
Quellcode (src/)	254
Frontend (frontend/)	255
Frequenzlisten (frequencylist/)	260
Konfigurationsdateien	260
Build-System	262
Dateiabhängigkeiten	262
Datenfluss	263
Leitfaden zum Ändern von Dateien	263
Wichtige Pfade	264
Übliche Dateioperationen	264
Versionsverwaltung	265

KAPITEL 1

README

README.md

PhantomSDR-Plus

Ein erweiterter Open-Source-WebSDR-Server mit fortschrittlichen Funktionen und futuristischem Design

PhantomSDR-Plus ist ein Fork von PhantomSDR und stellt einen leistungsstarken Software-Defined-Radio-(SDR-)Webserver bereit, der Hunderte gleichzeitiger Nutzer bedienen kann. Er bietet eine verbesserte Benutzeroberfläche, Unterstützung mehrerer Decoder, eine Bandplan-Visualisierung sowie Kompatibilität mit verschiedenen SDR-Hardwareplattformen.

Wichtigste Funktionen

Leistung und Skalierbarkeit

- **Mehrbenutzerbetrieb:** Hunderte gleichzeitiger Nutzer, je nach Hardware
- **Hohe Abtastrate:** Unterstützung von SDRs bis 70 MSPS (real) / 35 MSPS (IQ)
- **Hardwarebeschleunigung:** OpenCL- und CUDA-Unterstützung für GPU-beschleunigte Verarbeitung
- **Optimiertes Streaming:** FLAC- und Opus-Audiokompression mit geringer Latenz. Der Browser decodiert jedes FLAC-Paket in dem Moment, in dem es eintrifft, und sein Wiedergabepuffer passt sich der Verbindung des Hörers an und hält nur so viel Ton vorrätig wie nötig — typisch 100–150 ms vom Empfänger bis zum Lautsprecher auf einem Telefon im WLAN

Benutzeroberfläche

- **Futuristisches Design:** moderne, responsive Weboberfläche
- **Für Mobilgeräte optimiert:** verbesserte mobile Oberfläche für unterwegs
- **Bandplan-Visualisierung:** interaktiver Wasserfall mit Frequenzband-Overlays
- **Anpassbare Farbpaletten:** mehrere Farbschemata für den Wasserfall
- **Doppeltes S-Meter:** wahlweise analoge oder digitale Signalanzeige
- **Decoder-Tasten mit einem Druck:** eine Decoder-Reihe im Hauptfeld direkt unter der Modus-Auswahl (FT8, FT4, FT2, JS8, CW, WSPR, FAX, SSTV, NAVTX, RTTY), dazu RADEL und RADEU neben der Modus-Auswahl selbst sowie in den Pop-ups für Modi und Bänder. Ein Druck startet den Decoder und öffnet sein Fenster, ein zweiter beendet ihn. Die bisherige Bandbreiten-Reihe ist entfallen.
- **Kanalsuchlauf:** durchsucht das Band oder genau den sichtbaren Wasserfallausschnitt und hält beim ersten Kanal an, der eine wählbare Anzahl dB über dem Rauschflur des Bandes liegt; sobald der Kanal wieder still ist, läuft er von selbst weiter. Die Schrittweite folgt der Betriebsart, und die Stopps liegen auf dem Kanalaraster (1 kHz SSB, 100 Hz CW, 5 kHz Kurzwellen-AM, 9/10 kHz Mittelwelle, 9 kHz Langwelle). Ein Modus *Leere überspringen* liest das Wasserfallspektrum und springt direkt zu den Signalen, und eine Sperrtaste nimmt einen dauerbelegten Kanal aus dem Suchlauf. Alles läuft lokal im Browser des Hörers, der Suchlauf verstellt den Empfänger für niemanden sonst (siehe [Benutzerhandbuch](#))
- **Empfangsdiversität:** koppelt den Empfänger mit einem zweiten an einem anderen Ort und gibt jeweils die Seite mit dem besseren Signal wieder, sodass ein Schwund an der einen Station von der anderen aufgefangen wird. Der Partner kann ein weiterer PhantomSDR-Plus, ein KiwiSDR, ein

UberSDR oder ein WebSDR sein; die ersten drei brauchen nichts außer dem Browser des Hörers, und nur ein WebSDR benötigt ein kleines Relais auf Ihrem eigenen Server. Die Ausrichtung dauert etwa fünfzehn Sekunden, und ein Fernabgleich des SNR gleicht die beiden Stationen aneinander an. Jeder Hörer führt im eigenen Browser eine benannte Liste zweiter Empfänger in selbst gewählter Reihenfolge und kann sie in eine Datei exportieren und wieder einlesen (siehe [Empfangsdiversität](#))

- **Mehrere Empfänger, eine Seite:** Ein Rechner kann zwei oder mehr Empfänger gleichzeitig betreiben — etwa einen RX-888 für Kurzwelle und einen RTL-SDR für 2 m — und die Hörer wechseln mit Schaltflächen im Seitenkopf zwischen ihnen. Jeder Empfänger behält seinen eigenen Wasserfall, Chat, Markierungen, Hörerliste und Stationsangaben, und oberhalb von 30 MHz zeigt das S-Meter VHF-S-Stufen ($S9 = -93 \text{ dBm}$) (siehe [Mehrere Empfänger](#))

Signalverarbeitung

- **Mehrere Demodulationsarten:** AM, FM, USB, LSB, CW und mehr. Ein RADE-Decoder der Version 1 wurde ebenfalls implementiert.
- **Synchrone AM-Demodulation:** verbesserte AM-Empfangsqualität
- **Rauschminderung:** NR (spektral), NB (Störaustaster), NS (Grundrauschunterdrückung) und AN (automatische Notch) — sämtlich am Hörweg, nie an den Decodern
- **AGC-Optionen:** mehrere Betriebsarten der automatischen Verstärkungsregelung
- **Auto-Squelch:** automatische, rauschbasierte Rauschsperrschwelle
- **CTCSS-Subton-Rauschsperr:** FM-Audio ist nur zu hören, solange der gewählte Subton — oder ein beliebiger Standard-Subton — empfangen wird; der Ton wird in einem kleinen Fenster neben der CTCSS-Taste gewählt, die ihn dann anzeigt
- **KI-Rauschunterdrückung:** neuronales Netz RNNoise im Browser des Hörers, entfernt Bandrauschen aus Sprache (nur Sprachbetriebsarten; siehe [Benutzerhandbuch](#))

Erweiterte Funktionen

- **Digitale Decoder:** FT8, FT4, FT2, JS8, CW, QRSS Grabber, WSPR, HF-FAX, SSTV, NAVTEX, FSK/RTTY, PSK31, Olivia und FreeDV RADE, jeweils in einem eigenen Hintergrund-Thread, sodass die Dekodierung das Audio nie unterbricht (siehe [Decoder](#))
- **Decoder ID:** benennt die digitale Betriebsart im Durchlassbereich und bietet mit einem Klick den passenden Decoder an — aus gemessener Bandbreite, Tonabstand, Symbolrate und Burst-Timing, wobei die Frequenz als zusätzlicher Hinweis einfließt, was zugleich **FT8 von JS8** trennt, zwei Betriebsarten mit identischem Signal. Es schlägt nur vor, schaltet nie von selbst um, und schweigt lieber, als bei zu schwachem Signal zu raten. Schwund bringt es nicht zum Verstummen: Zerlegt QSB eine zweiminütige WSPR-Aussendung in Bruchstücke, benennt es die Betriebsart weiterhin, liest den Takt aus dem gesamten Beobachtungszeitraum und senkt die angezeigte Sicherheit, um den schwächeren Beleg kenntlich zu machen. Zwischen etwa 400 Hz und 2700 Hz spielt die Lage im Durchlassbereich keine Rolle; außerhalb erscheint eine bernsteinfarbene Schaltfläche **Recentre & retry**, die die Abstimmung einmal verschiebt und neu misst. Nehmen Sie einen Vorschlag an, wird der Empfänger auf diese Betriebsart eingestellt: Die Abstimmfrequenz rückt in die Mitte des Wasserfalls, die Ansicht wird rund 100 kHz breit, und Seitenband und Durchlassbereich werden die, mit denen diese Betriebsart gearbeitet wird — das 3-kHz-Teilband für die FT8-Familie, 1350–1650 Hz für WSPR, $\pm 250 \text{ Hz}$ und CW für Morsetelegrafie, und so weiter. Standardmäßig aus, im Betrieb etwa 0,5 % eines Kerns (siehe [Decoder](#))
- **Voreinstellungen für digitale Betriebsarten:** FT8, FT4, FT2, JS8 oder WSPR setzen Seitenband und Durchlassbereich zugleich
- **JS8-Gesprächsfenster:** Nachrichten aus mehreren Frames werden zu ganzen Sätzen zusammengesetzt
- **Statistik-Dashboard:** Server- und Nutzerstatistiken in Echtzeit

- **Liste der verbundenen Nutzer:** alle, die gerade zuhören, mit einer Abstimm Schaltfläche in jeder Zeile. Ihre eigene Sitzung ist mit **you** markiert — sowohl im Fenster der Oberfläche als auch, wenn `users.html` als eigene Seite geöffnet wird
- **Automatisches Spot-Reporting:** Ein Autorun-Daemon dekodiert FT8, FT4 und WSPR auf dem Server und lädt Spots zu PSK Reporter und WSPRnet hoch, mit zwei Zählern im Admin-Panel — Kacheln je Decoder für den aktuellen Lauf und ein Gesamtwert seit Beginn neben jedem Band-/ Betriebsart-Kontrollkästchen
- **Systemgraphen:** eine Seite **Graphen** im Admin-Panel mit CPU-Takt, CPU-Last, CPU-Temperatur und Nutzern online über die letzten 15 Minuten bis 24 Stunden (siehe [Admin-Panel](#))
- **Thermischer Schutz:** ein Wächter im Admin-Panel stoppt den Server bei CPU-Überhitzung und startet ihn wieder, sobald sie abgekühlt ist — mit Schwellen aus dem kritischen Grenzwert Ihrer eigenen CPU und ohne Annahmen darüber, wie Sie den Server starten oder stoppen. Ausgeliefert im reinen Protokollmodus, er tut also nichts, bis Sie ihn aktivieren (siehe [Thermal Guard](#)) Für einen unbeaufsichtigten Empfänger wird **dringend empfohlen**, das Panel als systemd-Unit zu betreiben — der Wächter läuft darin, also nimmt ein Neustart ihm sonst den Schutz (siehe [Admin-Panel](#)).
- **Emulation von KiwiSDR-Clients:** eine optionale Brücke, die auf demselben Host und Port zusätzlich das KiwiSDR-Protokoll beantwortet, sodass Kiwi-Software wie **AetherSDR** und `kiwiclient` sich direkt mit dem Empfänger verbindet — mit echtem Abstimmen, Wasserfall und einem S-Meter auf derselben Skala wie das Web-S-Meter. Aus, bis `[kiwi_emulation] enabled = true` in Ihrer Konfiguration steht (siehe [Installation](#))
- **Transceiver-Steuerung (CAT):** hält Ihr eigenes Funkgerät und Ihr Empfängerfenster auf derselben Frequenz, Betriebsart und Filterbreite, in eine Richtung oder in beide – über das Menü Rig von [Desktop PhantomSDR+](#) (Icom, Yaesu, Kenwood, Elecraft, FlexRadio und QRP Labs direkt, jedes andere Funkgerät über Hamlib, oder flrig) oder mit dem CATsync Tool for WebSDRs im Browser. Die Seiten stellen `catsync_*`-Funktionen für Frequenz, Betriebsart, Filterbreite und Stummschaltung bereit. Es funktioniert mit Empfängern PhantomSDR-Plus, KiwiSDR, PA3FWM WebSDR und UberSDR (siehe [Transceiver-Steuerung](#))
- **Lesezeichensystem:** Import/Export von Frequenz-Lesezeichen
- **Tastenkürzel:** effiziente Navigation und Bedienung
- **Mausrad-Unterstützung:** intuitive Frequenzabstimmung
- **WebSDR-Verzeichnis:** Anbindung an <https://sdr-list.xyz>

Unterstützte Hardware

PhantomSDR-Plus unterstützt eine breite Palette von SDR-Empfängern:

Gerät	Abtastrate	Format	Schnittstelle
RX888 MK2	bis 64 MHz	16 Bit	nativ
RTL-SDR	bis 3.2 MHz	8 Bit	rtl_sdr
HackRF One	bis 20 MHz	8 Bit	hackrf_transfer
Airspy HF+ / Discovery	bis 912 kHz	16 Bit	SoapySDR (SoapyAirspyHF) + rx_sdr
SDRplay RSP1A	bis 10 MHz	16 Bit	SoapySDR (libmirisdr-5 + SoapyMiri oder die SDRplay-API) + rx_sdr
RigExpert Fobos SDR	50 MHz (HF-Direktabtastung) / bis 20 MHz (RF)	16 Bit / 32-Bit-Float	SoapySDR (SoapyFobosSDR) + rx_sdr
Weitere Geräte	variabel	verschieden	SoapySDR/rx_tools

Schnellstart

Systemanforderungen

Minimum:

- Ubuntu 22.04 LTS (empfohlen) oder Fedora
- 2-Kern-CPU
- 4 GB RAM
- 10 GB Festplattenspeicher

Empfohlen:

- Ubuntu 24.04 LTS
- CPU mit 4+ Kernen (Ryzen 5 2600 oder Intel i5-6500T oder besser)
- 8 GB RAM
- GPU mit OpenCL-Unterstützung (optional, aber sehr empfehlenswert)
- SSD-Speicher

Installation

```
# Das Repository klonen
git clone --recursive https://github.com/svlbtl/PhantomSDR-Plus
cd PhantomSDR-Plus

# Skripte ausführbar machen
chmod +x *.sh

# Automatische Installation ausführen
./install.sh
```

Hinweis: Starten Sie nach dem Ausführen von `install.sh` Ihr Terminal neu, bevor Sie fortfahren.

Läuft PhantomSDR-Plus bei Ihnen schon? Dann nicht neu installieren, sondern aktualisieren. Holen Sie das Update-Werkzeug einmal und starten Sie es; Konfiguration, Marker, Admin-Passwort, Frequenzliste und Chatverlauf werden nie angerührt, und was Sie selbst geändert haben, wird Ihnen vorgelegt statt überschrieben:

```
bash cd ~/PhantomSDR-Plus curl -fLO https://raw.githubusercontent.com/svlbtl/PhantomSDR-Plus/main/update.sh chmod +x update.sh ./update.sh
```

Die letzte Zeile *meldet* nur, was sich ändern würde, und schreibt nichts; `./update.sh --apply` führt es aus. Einzelheiten im Kapitel *PhantomSDR-Plus aktualisieren*. Den Installer über eine laufende Station zu wiederholen ist nur nötig, wenn der Neubau an fehlenden Systempaketen scheitert.

Nehmen Sie `install_fedora.sh` auf Fedora, `install_arch.sh` auf Arch oder `install_opensuse.sh` auf openSUSE Tumbleweed — sie leisten dieselbe Arbeit mit einem anderen Paketmanager. Jede Debian- und Ubuntu-Version deckt `install.sh` selbst ab.

Nichts muss vorher von Hand installiert werden. Das Skript installiert die Build-Abhängigkeiten und Node.js, baut das Backend, baut den Treiber für den von Ihnen gewählten Empfänger (beim RX888 samt udev-Regeln), öffnet `site_information.json` zum Ausfüllen, baut das Frontend, installiert OpenCL, wenn die Hardware es unterstützt, und richtet dann das **Admin-Panel**, den **FreeDV-RADE-Decoder** und den **Statistikserver** ein — alle drei standardmäßig, mit `n` überspringen — und schließt mit einem vollständigen `recompile.sh` ab. Es ist durchgehend interaktiv und dauert von zwanzig Minuten bis über eine Stunde.

Die Schritt-für-Schritt-Beschreibung finden Sie in [INSTALLATION.md](#) → [Was das Installationsskript erledigt](#). Ausführliche Installationsanweisungen finden Sie in [INSTALLATION.md](#).

Leistungsmessungen

Hardware	Abtaste	CPU-Auslastung	Nutzerkapazität
Ryzen 5 2600 (alle Kerne)	64 MHz (32 MHz IQ)	38-40 %	100+ Nutzer
AMD RX 580 (GPU)	64 MHz (32 MHz IQ)	28-35 %	100+ Nutzer
Intel i5-6500T (mit OpenCL)	60 MHz (30 MHz IQ)	10-12 %	100+ Nutzer

Hinweis: Der CPU-Mehraufwand pro Nutzer ist minimal (< 1 % pro Nutzer), wenn die Hardwarebeschleunigung aktiviert ist.

Verwendung

Grundlegender Betrieb

- SDR konfigurieren:** Bearbeiten Sie die passende Konfigurationsdatei (z. B. `config-rtl.toml`)
- Standortinformationen aktualisieren:** Bearbeiten Sie `frontend/site_information.json`
- Server starten:** Führen Sie das passende Startskript aus: `bash ./start-rtl.sh` # Für RTL-SDR `./start-rsp1a.sh` # Für SDRplay RSP1A `./start-airspyhf.sh` # Für Airspy HF+ `./start-fobos-hf.sh` # Für RigExpert Fobos SDR, HF1/HF2 (0-25 MHz) `./start-fobos.sh` # Für RigExpert Fobos SDR, RF (25-6000 MHz) `./start-hackrf.sh` # Für HackRF One `./start-rx888mk2.sh` # Für RX888 MK2 RSP1A, Airspy HF+ und Fobos laufen über SoapySDR und `rx_sdr`; das Installationsskript richtet ihren Treiber ein, wenn Sie sie wählen (Optionen 3, 6 und 5), oder Sie starten auf einer installierten Station `./setup-rsp1a.sh`, `./setup-airspyhf.sh` oder `./setup-fobos.sh`. Siehe [INSTALLATION.md](#). Der HackRF

One (Option 7) braucht nur das Paket `hackrf` der Distribution: `./setup-hackrf.sh`. Jedes Startskript ist **eigenständig**: Es beendet eine laufende Instanz, startet den Empfänger + `spectrumserver`, **löst sich in den Hintergrund** (übersteht das Schließen des Terminals) und betreibt einen **Watchdog**, der die Kette bei einem Absturz automatisch neu startet. Der Fortschritt wird in `logwebsdr.txt` protokolliert — beobachten Sie ihn mit `tail -f logwebsdr.txt`. Ein erneuter Aufruf eines Startskripts entspricht einem sauberen Neustart, und eine `flock`-Sperrung stellt sicher, dass immer nur ein Empfänger läuft. Optional: `SPECTRUM_CORES=0-3`, um CPUs festzulegen, `RX_ARGS=""`, um Empfängerargumente ohne Bearbeiten der Datei zu überschreiben (`RX888_ARGS` beim RX888), `RX_DRIVER=miri|sdrplay`, um den RSP1A-Treiber zu wählen.

4. **Oberfläche aufrufen**: Öffnen Sie im Browser `http://localhost:PORT` (Standardport je nach Konfiguration unterschiedlich)

Server stoppen

Ein gemeinsames Stopp-Skript funktioniert für **jeden** Empfänger — es beendet zuerst den Watchdog (damit er nicht automatisch neu startet), dann den Empfänger und `spectrumserver`:

```
./stop-websdr.sh
```

Konfiguration

Wesentliche Konfigurationsdateien

1. `config-[Gerät].toml` – Server- und SDR-Konfiguration
2. Servereinstellungen (Port, Threads, HTML-Wurzelverzeichnis)
3. Eingangseinstellungen (Abtastrate, FFT-Größe, Frequenz)
4. Optionen zur WebSDR-Registrierung
5. Einstellungen zur Audiokompression
6. Nach Änderungen den Server neu starten.
7. `frontend/site_information.json` – öffentliche Standortinformationen
8. Angaben zum Betreiber (Rufzeichen, E-Mail, Standort)
9. Informationen zu Hardware und Antenne
10. Region- und Bandbreiteneinstellungen
11. Chat aktivieren/deaktivieren
12. Führen Sie nach Änderungen `recompile.sh` in einem Terminal aus.
13. `markers.json` – Frequenzmarker und Bandplan
14. `frontend/src/bands-config.js` – Bandplan-Konfiguration Diese Datei liegt unter `frontend/src/bands-config.js` und definiert die Bänder, die der SysOp anlegt.
15. Sie sehen etwa Folgendes und können es nach Belieben bearbeiten: ``
16. `const bands = .....`
17. `{ ITU: 1, name: '40m', min: -30, max: 110, initFreq: '7120000', publishBand: '1', startFreq: 7000000, endFreq: 7200000, stepi: 1000, color: 'rgba(50, 168, 72, 0.6)', modes: [{ mode: MODES.CW, startFreq: 7000000, endFreq: 7040000 }, { mode: MODES.LSB, startFreq: 7040000, endFreq: 7200000 }] }, { ITU: 2, name: '40m', min: -30, max: 110, initFreq: '', publishBand: '1', startFreq: 7000000, endFreq: 7300000, stepi: 1000, color: 'rgba(50, 168, 72, 0.6)', modes: [{ mode: MODES.CW, startFreq: 7000000, endFreq: 7050000 }, { mode: MODES.LSB, startFreq: 7050000,`

```
endFreq: 7300000 } ] }, { ITU: 3, name: '40m', min: -30, max: 110, initFreq: '', publishBand: '1',
startFreq: 7000000, endFreq: 7200000, stepi: 1000, color: 'rgba(50, 168, 72, 0.6)', modes: [ { mode:
MODES.CW, startFreq: 7000000, endFreq: 7040000 }, { mode: MODES.LSB, startFreq: 7040000,
endFreq: 7200000 } ] ..... etc ``` Dabei gilt:
```

18. **ITU** ist die Region, in der sich der Server befindet,
19. **min und max** sind die Helligkeitsgrenzen des Wasserfalls für das jeweilige Band,
20. **initFreq** ist die Frequenz, auf die bei Auswahl des Bandes zunächst abgestimmt wird,
21. **publishBand** legt fest, ob das Band angezeigt wird: 1 für Amateurbänder, 2 für Rundfunkbänder,
22. **startFreq und endFreq** definieren die Bandgrenzen,
23. **stepi** ist die Standard-Mausradschrittweite für das Band, und **modes** ist die bevorzugte Standardbetriebsart für das Band.
24. Führen Sie nach Änderungen **recompile.sh in einem Terminal** aus dem Ordner PhantomSDR-Plus aus.

Konfigurationsbeispiele finden Sie in den mitgelieferten Beispieldateien:

- `config-rtl.toml` – RTL-SDR-Konfiguration
- `config-rspla.toml` – SDRplay-RSP1A-Konfiguration
- `config-airspyhf.toml` – Airspy-HF+-Konfiguration
- `config-fobos-hf.toml` / `config-fobos.toml` – Konfiguration für RigExpert Fobos SDR (HF / RF)
- `config-hackrf.toml` – HackRF One
- `config-rx888mk2.toml` – RX888-MK2-Konfiguration

Wasserfall-Untergrenze — `waterfall.sh`

`waterfall.sh` (im Wurzelverzeichnis des Projekts) ändert den **voreingestellten minimalen Wasserfallpegel** in dB — also den Wert, bei dem der Wasserfall bei einem neuen Besuch beginnt.

Dieser Wert steht in zwei Quelldateien an fünf verschiedenen Stellen (`frontend/src/waterfall.js` → `this.minWaterfall` sowie `frontend/src/App.svelte` → das anfängliche `let min_waterfall`, die beiden „min“-Rücksetzfälle und der Lesezeichen-Standardwert). Von Hand ist das leicht falsch gemacht, deshalb erledigt es das Skript für Sie. Es verwendet niemals fest verdrahtete Zeilennummern — jede Stelle wird über ein Muster gefunden, sodass es auch nach Aktualisierungen der Quellen weiter funktioniert.

- Ein **höherer** Wert (z. B. `-15`) ergibt einen **dunkleren** Wasserfall.
- Ein **niedrigerer** Wert (z. B. `-45`) ergibt einen **helleren** Wasserfall.

```
./waterfall.sh          # interaktiv – zeigt die aktuellen Werte und fragt nach dem neuen
./waterfall.sh -v -15   # Wert ohne Rückfrage setzen
./waterfall.sh -v -15 -y # ... und alle Bestätigungen überspringen
./waterfall.sh -s       # nur die aktuellen Werte anzeigen, nichts ändern
```

Vor dem Ändern legt es von beiden Dateien eine Sicherung mit Zeitstempel an (`*.bak-YYYYmmdd-HHMMSS`) und gibt den Befehl zum Zurückspielen aus. Angefasst werden nur Literale, die dem gerade verwendeten Wert entsprechen, sodass unbeteiligte Zahlen in diesen Dateien niemals versehentlich überschrieben werden können. Erscheint der neue Wert danach nicht, stellt das Skript die Sicherungen selbst wieder

her und bricht mit einem Fehler ab. Es warnt außerdem, wenn die beiden Dateien nicht übereinstimmen, und setzt beide.

- Nach der Änderung muss das Frontend neu gebaut werden — das Skript bietet an, `recompile.sh` für Sie auszuführen (mit `-y` überspringt es den Neubau, führen Sie dann `./recompile.sh` selbst aus, wenn Sie so weit sind).

Anpassung

Hintergrundbild ändern

Ersetzen Sie `frontend/src/assets/background.jpg` durch Ihr bevorzugtes Bild (behalten Sie denselben Dateinamen bei).

- Führen Sie nach der Änderung `recompile.sh` in einem Terminal aus.

Auswahl des Audio-Codecs

Wählen Sie in Ihrer `.toml`-Konfiguration zwischen FLAC und Opus:

```
[input]
audio_compression="opus" # or "flac"
```

- Starten Sie nach der Änderung den Server neu.

S-Meter-Kalibrierung

Die beiden Instrumente laufen absichtlich über **getrennte Ketten** und werden daher mit zwei verschiedenen Stellgrößen abgeglichen.

Die zwei Stellgrößen

Beide stehen im Abschnitt `[input]` der Konfigurationsdatei, mit der der Server gestartet wird (`config.toml`, `config-rx888mk2.toml` usw.):

```
[input]
smeter_offset=5          # nur der digitale Balken
analog_smeter_offset=5   # nur der analoge Zeiger
```

Sie werden in `src/websocket.cpp` gelesen und im ersten `basic_info`-Frame an den Browser geschickt.

Ein Neustart des Servers genügt also — ein Neubau des Frontends ist nicht nötig.

Analoge Kette

```
angezeigte dBm = -130 + (rawDb + analog_smeter_offset + 130) x 1,1
```

- Die visuelle Verstärkung von 1,1 wirkt *nach* dem Offset, mit Drehpunkt bei -130 dBm. Eine Einheit `analog_smeter_offset` verschiebt die Anzeige also um 1,1 dB. Für eine Verschiebung um X dB gilt `analog_smeter_offset ≈ 0,91 x X`. Der TOML-Wert wird als Ganzzahl gelesen, es sind also nur ganze Schritte möglich.
- Die Zeigerkennlinie (`powerFromDbm` in `frontend/src/lib/SMeterAnalog.svelte`) legt **S9 auf -73 dBm** bei 60 von 100 Zeigereinheiten, mit einer `pow(...,0,6)`-Kurve von -130 bis -73 unterhalb S9 und einer `pow(...,0,8)`-Kurve von -73 bis -13 (S9+60) darüber.
- Das ist **keine** lineare Skala mit 6 dB pro S-Stufe. Der sinnvolle Bezugspunkt ist deshalb S9 — die Marken unterhalb S9 folgen einem Messsender nicht in 6-dB-Schritten.

Digitale Kette

```
value    = (rawDb / 150) x 100 + smeter_offset    -> begrenzt auf [-100, 0]
segments = round((value + 100) x 35 / 100) + DIGITAL_BAR_TRIM
```

- Man beachte das $/150 \times 100$: die digitale Skala ist **auf 2/3 gestaucht** und sieht weder `analog_smeter_offset` noch die 1,1-fache visuelle Verstärkung. Deshalb stimmen Balken und Zeiger nie exakt überein — das ist so gewollt.
- 35 Segmente auf 100 Einheiten bedeuten: ein Segment sind 2,86 Offset-Einheiten, also etwa 4,3 dB roh. `smeter_offset=3` entspricht somit ungefähr einem Segment.
- `DIGITAL_BAR_TRIM` in `frontend/src/lib/SMeterDigital.svelte` (derzeit 0) verschiebt den Balken um ganze Segmente. Eine Änderung ist ein Eingriff in den Quelltext und **erfordert** einen Neubau des Frontends.

Die LED-Fenster werden in beiden Instrumenten analog gespeist

Die Fenster dBm / dBμV / SNR / NF lesen in **beiden** Instrumenten den analog kalibrierten Wert und addieren einen festen `VISUAL_DBM_OFFSET` von 5, fest verdrahtet in `SMeterAnalog.svelte` und `SMeterDigital.svelte`. Daraus folgt:

- `smeter_offset` ändert **keine** Zahl in den Fenstern, nur die Balkenlänge.
- $\text{dB}\mu\text{V} = \text{dBm} + 107$ (50 Ohm), und $\text{NF} = \text{dBm} - \text{SNR}$ per Konstruktion, NF folgt also automatisch.
- Sollen die **Zahlen** stimmen, ohne die Zeigerstellung anzutasten, ist `VISUAL_DBM_OFFSET` die richtige Stellgröße — aber es ist eine Konstante im Quelltext, erfordert einen Neubau und muss in **beiden** Dateien gleich geändert werden.

Vorgehen

1. Einen bekannten Pegel in den Antenneneingang einspeisen (Messsender mit -73 dBm = S9), in SSB/ CW mit stabiler Durchlassbreite. Die Anzeige hängt von Bandbreite und AGC ab, also zuerst Modus und Bandbreite festlegen.
2. Ablesen, was das analoge dBm-Fenster zeigt, dann `analog_smeter_offset` $\approx 0,91 \times (-73 - \text{Anzeige})$ setzen. Server neu starten und erneut prüfen. Ein Durchgang sollte genügen; der Zeiger sollte auf S9 stehen.
3. Auf das digitale Instrument umschalten und `smeter_offset` so setzen, dass die Balkenlänge S9 dorthin legt, wo man es haben will — etwa 3 Einheiten pro Segment. Dieser Schritt ist reine Optik und beeinflusst die Zahlenwerte nicht.
4. Ohne Messsender ist ein ruhiges Band mit einer Bake bekannten Pegels ein brauchbarer Ersatz, oder man legt schlicht das Grundrauschen auf einen plausiblen Wert (z. B. -120 dBm auf 20 m mit ordentlicher Antenne) — wohl wissend, dass damit die gesamte Kette einschließlich Antennengewinn kalibriert wird, nicht nur der Empfänger.

GUI-Varianten

Alle vier GUI-Varianten (analoges/digitales S-Meter x Layout v1/v2) stecken in einem einzigen Build. Besucher wechseln über das Menü oben rechts zwischen ihnen — nichts wird neu geladen, Audio, Wasserfall und laufende Decoder machen weiter — und die Wahl wird pro Browser gespeichert.

- Führen Sie `recompile.sh` in einem Terminal aus, um die *Startvariante* festzulegen, also das, was ein Erstbesucher sieht.
- **EDITING_VARIANTS.md** – wie man die Frontend-Varianten (S-Meter und Layout) bearbeitet und neu baut

Dokumentation

- **INSTALLATION.md** - vollständige Installationsanleitung für Systembetreiber
 - **ADMIN_PANEL_SETUP.md** - Installationsanleitung für administrierende Systembetreiber
 - **USER_GUIDE.md** - Anleitung für Endnutzer zur Bedienung des WebSDR
 - **THERMAL_GUARD.md** - Sysop-Handbuch für den CPU-Überhitzungsschutz: die vier Modi und was bei jedem zu tun ist
 - **CONNECTION_LIMITS.md** - Sysop-Handbuch für Verbindungslimits: einen öffentlichen Empfänger vor Verbindungsfluten schützen
 - **MULTI_RECEIVER.md** - Sysop-Handbuch für mehrere Empfänger auf einem Rechner: Instanzen, `receivers.toml`, die Empfängerauswahl und `start-all.sh`
 - **PROJECT_STRUCTURE.md** - Verzeichnisstruktur und Codeorganisation
 - **EDITING_VARIANTS.md** - wie man die Frontend-Varianten (S-Meter und Layout) bearbeitet und neu baut
-

Online-WebSDR-Liste

Registrieren Sie Ihren WebSDR im offiziellen Verzeichnis: <https://sdr-list.xyz>

Setzen Sie `register_online=true` in Ihrer `.toml`-Konfigurationsdatei, um sich automatisch zu registrieren.

Fehlerbehebung

Häufige Probleme

OpenCL funktioniert nicht

- Stellen Sie sicher, dass die Treiber korrekt installiert sind
- Prüfen Sie es mit dem Befehl `clinfo`
- Siehe `INSTALLATION.md` für die ausführliche OpenCL-Einrichtung

Probleme mit der Audiolatenz

- Bleiben Sie bei FLAC als Codec: Es ist der mit der geringeren Latenz, während Opus pro Paket bis zu 20 ms zurückhält
- Öffnen Sie die Seite mit angehängtem `?diag=1` (zum Beispiel `/mobile?diag=1`), um die Verzögerung aufgeteilt in Paket, Puffer und Ausgabe zu sehen; der Puffer schrumpft innerhalb von ein, zwei Minuten nach einer Netzstörung von selbst
- Bricht der Ton ab, erhöhen Sie die Pufferstufe um eins (siehe Benutzerhandbuch, *Audiopuffer und Verzögerung*)
- Sorgen Sie für ausreichende CPU-/GPU-Ressourcen

Build-Fehler

- Prüfen Sie, ob alle Abhängigkeiten installiert sind
 - Versuchen Sie, das Build-Verzeichnis zu bereinigen: `rm -rf build && meson setup build`
 - Prüfen Sie auf widersprüchliche Bibliotheksversionen
-

Mitwirken

Beiträge sind willkommen! Dies ist ein unabhängiger Fork mit zusätzlichen Funktionen. Bitte:

1. Forken Sie das Repository
 2. Erstellen Sie einen Feature-Branch
 3. Committen Sie Ihre Änderungen
 4. Pushen Sie den Branch
 5. Erstellen Sie einen Pull Request
-

Lizenz

Dieses Projekt steht unter der GNU General Public License v3.0 – Einzelheiten finden Sie in der Datei [LICENSE](#).

Autoren und Danksagungen

- **SV1BTL und SV2AMK** – Entwicklung und Erweiterungen von PhantomSDR-Plus
 - Basierend auf dem ursprünglichen PhantomSDR-Projekt
-

Links

- **Live-Demo:** <http://phantomsdr.no-ip.org:8900/>
 - **WebSDR-Verzeichnis:** <https://sdr-list.xyz>
 - **GitHub-Repository:** <https://github.com/sv1btl/PhantomSDR-Plus>
-

Unterstützung

- **Probleme:** Melden Sie Fehler über [GitHub Issues](#)
 - **E-Mail:** Kontakt sv1btl@otenet.gr
-

Leistungstipps

1. **Aktivieren Sie OpenCL/CUDA** für GPU-Beschleunigung – reduziert die CPU-Last drastisch
 2. **Verwenden Sie SSD-Speicher** für bessere E/A-Leistung
 3. **Weisen Sie ausreichend RAM zu** – mindestens 8 GB empfohlen
 4. **Optimieren Sie die FFT-Größe** – größere FFT = bessere Auflösung, aber mehr CPU-Last
 5. **Erwägen Sie eine dedizierte GPU** – AMD oder NVIDIA mit OpenCL-Unterstützung
-

73 de SV1BTL & SV2AMK

Ausführliche Einrichtungsanweisungen finden Sie in *INSTALLATION.md* Die Bedienungsanleitung für Endnutzer finden Sie in *USER_GUIDE.md*

KAPITEL 2

Installationsanleitung

INSTALLATION.md

PhantomSDR-Plus Installationsanleitung für Systembetreiber

Diese ausführliche Anleitung führt Sie durch Installation und Konfiguration von PhantomSDR-Plus auf Ihrem Server.

Inhaltsverzeichnis

1. Systemanforderungen
2. Vorbereitung vor der Installation
3. PhantomSDR-Plus installieren — das Skript und [was es erledigt](#)
4. Autorun-Spot-Reporter (FT8/FT4/WSPR)
5. Emulation von KiwiSDR-Clients (optional)
6. Konfiguration
7. Gerätespezifische SDR-Einrichtung
8. Test und Überprüfung
9. Automatischen Start einrichten
10. CPU-Überhitzungsschutz
11. Fehlerbehebung

Nur als Nachschlagewerk — das Installationsskript erledigt all dies bereits für Sie. Lesen Sie diese Abschnitte, wenn Sie eine Distribution verwenden, die keines der Skripte abdeckt, oder wenn Sie einen einzelnen Schritt von Hand reparieren müssen:

- [Abhängigkeiten installieren](#)
 - [Node.js und npm installieren](#)
 - [OpenCL installieren](#)
 - [Von Hand bauen](#)
 - [Opus-Audiocodec installieren](#)
-

Systemanforderungen

Unterstützte Betriebssysteme

Primär (empfohlen):

- Ubuntu 22.04 LTS (Jammy), 24.04 LTS (Noble) und 26.04 LTS (Resolute) — 24.04 empfohlen
- Debian 12 (Bookworm) und Debian 13 (Trixie)

Alternative:

- Fedora (aktuelle stabile Version)

- Arch Linux (Rolling Release)
- openSUSE Tumbleweed (nicht Leap — siehe Hinweis weiter unten)

Hardwareanforderungen

Mindestkonfiguration:

- CPU: Zweikernprozessor (2 GHz oder mehr)
- RAM: 4 GB
- Speicher: 10 GB freier Platz
- Netzwerk: 100-Mbit/s-Verbindung

Empfohlene Konfiguration:

- CPU: Vierkerner oder besser (Ryzen 5 2600, Intel i5-6500T oder besser)
- RAM: 8 GB oder mehr
- Speicher: SSD ab 20 GB
- GPU: AMD/NVIDIA mit OpenCL-Unterstützung (sehr empfehlenswert)
- Netzwerk: 1-Gbit/s-Verbindung

Hochleistungskonfiguration:

- CPU: 6 Kerne oder mehr (Ryzen 7, Intel i7 oder besser)
- RAM: 16 GB oder mehr
- Speicher: NVMe-SSD
- GPU: dedizierte GPU mit OpenCL-/CUDA-Unterstützung
- Netzwerk: 1 Gbit/s oder besser

Vorbereitung vor der Installation

1. System aktualisieren

```
# Ubuntu/Debian
sudo apt update
sudo apt upgrade -y
sudo reboot

# Fedora
sudo dnf update -y
sudo reboot
```

2. Ubuntu-Version prüfen

```
lsb_release -a
```

Die erwartete Ausgabe sollte zeigen: Ubuntu 24.04 LTS

3. Verfügbaren Speicherplatz prüfen

```
df -h
```

Stellen Sie sicher, dass in Ihrem Heimatverzeichnis mindestens 10 GB frei sind.

4. CPU-Informationen prüfen

```
lscpu
```

Notieren Sie die Anzahl der Kerne/Threads zur Optimierung der Konfiguration.

Abhängigkeiten installieren

WICHTIG

Für eine normale Installation brauchen Sie diesen Abschnitt nicht. `./install.sh` — bzw. das Installationsskript Ihrer Distribution — erledigt all das für Sie; siehe [Was das Installationsskript erledigt](#). Das Folgende ist eine Referenz für eine manuelle Einrichtung, für eine Distribution, die keines der Skripte abdeckt, oder um einen einzelnen Schritt von Hand zu reparieren.

Ubuntu 24.04 LTS

```
sudo apt install -y \
  build-essential \
  cmake \
  pkg-config \
  meson \
  libfftw3-dev \
  libwebsocketpp-dev \
  libflac++-dev \
  zlib1g-dev \
  libzstd-dev \
  libboost-all-dev \
  libopus-dev \
  libliquid-dev \
  git \
  psmisc \
  wget \
  curl
```

Fedora

```
sudo dnf install -y \
  g++ \
  meson \
  cmake \
  fftw3-devel \
  websocketpp-devel \
  flac-devel \
  zlib-devel \
  boost-devel \
  libzstd-devel \
  opus-devel \
  liquid-dsp-devel \
  git \
  psmisc \
  wget \
  curl
```

Installation überprüfen

```
# Check if key dependencies are installed
pkg-config --modversion fftw3
cmake --version
meson --version
```

Node.js und npm installieren

WICHTIG

Für eine normale Installation brauchen Sie diesen Abschnitt nicht. `./install.sh` — bzw. das Installationsskript Ihrer Distribution — erledigt all das für Sie; siehe [Was das Installationsskript erledigt](#). Das Folgende ist eine Referenz für eine manuelle Einrichtung, für eine Distribution, die keines der Skripte abdeckt, oder um einen einzelnen Schritt von Hand zu reparieren.

PhantomSDR-Plus benötigt Node.js, um das Frontend zu bauen. Wir verwenden dafür NVM (Node Version Manager).

1. NVM installieren

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.4/install.sh | bash
```

2. NVM laden

WICHTIG: Schließen Sie Ihr Terminal und öffnen Sie es erneut, oder führen Sie aus:

```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
```

3. NVM-Installation überprüfen

```
nvm --version
```

Erwartete Ausgabe: `0.40.4` oder ähnlich

4. Node.js installieren

```
# Install Node.js 22 - the version every install script requires
nvm install 22

# Make it the default, so systemd units and cron see it too
nvm alias default 22

# Verify installation
node --version
npm --version
```

5. Optional: weitere Node-Versionen installieren

```
# Install latest version
nvm install node

# Install specific version (if needed)
nvm install 22.22.3

# List installed versions
nvm list

# Use specific version
nvm use 22
```

OpenCL installieren (optional, aber empfohlen)

WICHTIG

Für eine normale Installation brauchen Sie diesen Abschnitt nicht. `./install.sh` — bzw. das Installationsskript Ihrer Distribution — erledigt all das für Sie; siehe [Was das Installationsskript erledigt](#). Das Folgende ist eine Referenz für eine manuelle Einrichtung, für eine Distribution, die keines der Skripte abdeckt, oder um einen einzelnen Schritt von Hand zu reparieren.

OpenCL verbessert die Leistung erheblich, indem es FFT-Berechnungen auf die GPU auslagert. Dieser Abschnitt behandelt integrierte Intel-Grafik. Für AMD-/NVIDIA-GPUs beachten Sie die Dokumentation des Herstellers.

Intel-CPU mit integrierter Grafik

1. OpenCL-Basiskomponenten installieren

```
sudo apt install -y \
  build-essential \
  cmake \
  pkg-config \
  meson \
  libfftw3-dev \
  libwebsocketpp-dev \
  libflac++-dev \
  zlib1g-dev \
  libzstd-dev \
  libboost-all-dev \
  libopus-dev \
  libliquid-dev \
  libclfft-dev \
  ocl-icd-opencl-dev \
  clinfo
```

2. Intel Compute Runtime herunterladen

```
mkdir -p ~/neo
cd ~/neo

wget https://github.com/intel/compute-runtime/releases/download/19.07.12410/intel-
gmmlib_18.4.1_amd64.deb
wget https://github.com/intel/compute-runtime/releases/download/19.07.12410/intel-igc-
core_18.50.1270_amd64.deb
wget https://github.com/intel/compute-runtime/releases/download/19.07.12410/intel-igc-
opencl_18.50.1270_amd64.deb
wget https://github.com/intel/compute-runtime/releases/download/19.07.12410/intel-
opencl_19.07.12410_amd64.deb
wget https://github.com/intel/compute-runtime/releases/download/19.07.12410/intel-
ocloc_19.07.12410_amd64.deb
```

3. Intel-OpenCL-Runtime installieren

```
sudo dpkg -i *.deb
```

Falls Abhängigkeitsfehler auftreten:

```
sudo apt --fix-broken install
```

4. OpenCL-ICD-Loader installieren

```
sudo apt update
sudo apt install intel-opencl-icd
```

Falls Fehler auftreten:

```
sudo apt --fix-broken install
sudo apt install intel-opencl-icd
```

5. OpenCL-Installation überprüfen

```
sudo clinfo
```

Sie sollten Informationen zu Ihrer OpenCL-Plattform und den Geräten sehen. Achten Sie auf:

- Anzahl der Plattformen: 1 (oder mehr)
- Plattformname: Intel(R) OpenCL (oder ähnlich)
- Gerätetyp: GPU oder CPU

6. Neu starten

```
sudo reboot
```

OpenCL auf AMD-GPU

Installieren Sie für AMD-GPUs ROCm:

```
# Add ROCm repository
wget -q -O - https://repo.radeon.com/rocm/rocm.gpg.key | sudo apt-key add -
echo 'deb [arch=amd64] https://repo.radeon.com/rocm/apt/debian/ ubuntu main' | sudo tee /etc/apt/sources.list.d/rocm.list

# Install ROCm
sudo apt update
sudo apt install rocm-opencl rocm-clinfo

# Add user to video group
sudo usermod -a -G video $USER

# Reboot
sudo reboot

# Verify
clinfo
```

OpenCL auf NVIDIA-GPU

Installieren Sie für NVIDIA-GPUs CUDA:

```
# Install NVIDIA drivers
sudo apt install nvidia-driver-525

# Install CUDA toolkit
wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64/cuda-keyring_1.0-1_all.deb
sudo dpkg -i cuda-keyring_1.0-1_all.deb
sudo apt update
sudo apt install cuda

# Reboot
sudo reboot

# Verify
nvidia-smi
clinfo
```

PhantomSDR-Plus installieren

Repository klonen, Skripte ausführbar machen, Installation starten

```
cd ~
git clone --recursive https://github.com/sv1btl/PhantomSDR-Plus
cd PhantomSDR-Plus
chmod +x *.sh
./install.sh
```

⚠ **WICHTIG:** Starten Sie nach dem Ende der Installation **Ihr Terminal neu** — das frisch installierte Node.js und Rust liegen erst danach in Ihrem PATH.

Läuft PhantomSDR-Plus bei Ihnen schon? Dann nicht neu installieren, sondern aktualisieren. Holen Sie das Update-Werkzeug einmal und starten Sie es; Konfiguration, Marker, Admin-Passwort, Frequenzliste und Chatverlauf werden nie angerührt, und was Sie selbst geändert haben, wird Ihnen vorgelegt statt überschrieben:

```
bash cd ~/PhantomSDR-Plus curl -fLO https://raw.githubusercontent.com/sv1btl/PhantomSDR-Plus/main/update.sh chmod +x update.sh ./update.sh
```

Die letzte Zeile *meldet* nur, was sich ändern würde, und schreibt nichts; `./update.sh --apply` führt es aus. Einzelheiten im Kapitel *PhantomSDR-Plus aktualisieren*. Den Installer über eine laufende Station zu wiederholen ist nur nötig, wenn der Neubau an fehlenden Systempaketen scheitert.

Nehmen Sie das Skript, das zu Ihrem System passt. Alle leisten dieselbe Arbeit und stellen dieselben Fragen; nur der Paketmanager unterscheidet sich:

System	Skript
Ubuntu 22.04 / 24.04 / 26.04, Debian 12 / 13	<code>./install.sh</code>
Fedora	<code>./install_fedora.sh</code>
Arch	<code>./install_arch.sh</code>
openSUSE Tumbleweed	<code>./install_opensuse.sh</code>

Ein Installationsskript für jede Debian- und Ubuntu-Version. `install.sh` liest `/etc/os-release` und die installierte Boost-Version und passt sich selbst an; die früheren `install_ubuntu22.sh`, `install-Deb12.sh` und `install_ubuntu26.sh` sind entfallen. Es setzt `DEBIAN_FRONTEND=noninteractive`, damit das mit `python3-matplotlib` hereinkommende `tzdata` den Lauf nicht anhalten kann, um nach Ihrer Zeitzone zu fragen — und dann die Antwort verschluckt, die für die nächste Frage gedacht war. Unter Jammy ruft es direkt `install_rade_ubuntu22.sh` auf, denn Jammys `python3-websockets` ist 10.1, RADE braucht aber 11.0 oder neuer. Es weiß, wo `intel-opencl-icd` in jeder Version liegt: in den Repositories bei 22.04 und 26.04, in `non-free` bei Debian 12 und in Intels eigenem Grafik-Repository bei 24.04 und Debian 13. Und wo Boost 1.87 oder neuer ist, ist der `websocketpp`-Header-Patch nicht mehr optional — das Skript prüft, dass er angekommen ist, und baut sonst gar nicht erst. Ein neuerer Compiler wird nie gebraucht: Der Standard-GCC akzeptiert `-std=c++23` auf allen fünf Versionen, installieren Sie dafür also kein `gcc-12`.

Die Installation läuft in 17 klar nummerierten Schritten ab, und jede Stelle, an der auf Sie gewartet wird, ist von einem  **IHRE EINGABE WIRD BENÖTIGT**-Rahmen eingefasst, damit eine Frage nicht im durchlaufenden Text untergeht. Die sieben Fragen werden vorab aufgelistet, bevor irgendetwas installiert wird. `PHANTOM_NONINTERACTIVE=1` beantwortet alle mit ihren Vorgaben; die `PHANTOM_*`-Variablen stehen im Kopf von `install.sh`.

Jeder Lauf schreibt install.txt. Wenn das Installationsskript fertig ist — oder auf halbem Weg abbricht — schreibt es einen Bericht nach `install.txt` im PhantomSDR-Plus-Verzeichnis: das Ergebnis, jeden der 17 Schritte als OK / SKIPPED / PARTIAL / FAILED, was erkannt wurde (Distribution, Boost, Compiler, Node.js), welche Komponenten installiert wurden, und jede aufgetretene Warnung. Ein fehlgeschlagener Lauf hinterlässt einen Bericht, der beim gescheiterten Schritt endet, mit dem Grund und dem Hinweis, dass das Skript gefahrlos erneut laufen darf. Es ist die erste Datei, die man liest, wenn etwas nicht funktioniert hat, und die erste, die man an einen Fehlerbericht hängt. Jeder Lauf überschreibt sie — heben Sie eine Kopie auf, wenn Sie zwei Installationen vergleichen wollen.

Ubuntu 26.04, Arch und openSUSE Tumbleweed lassen sich übersetzen, sind aber noch nicht im Funkbetrieb erprobt. Alle drei bringen ein Boost neuer als 1.87, das die `io_service`-API entfernt hat, für die das mitgelieferte `websocketpp` 0.8.2 geschrieben wurde. Die gepatchten Header, die das Installationsskript hineinkopiert, überbrücken das (`io_context`, `executor_work_guard`, `boost::asio::post`, der moderne Resolver), und wo Boost 1.87 oder neuer ist, behandelt das Skript den Patch als zwingend statt optional — es prüft, dass die Kopien angekommen sind, und baut sonst nicht. Eine vollständige Installation samt Empfängertreiber und allen optionalen Komponenten wurde in Containern auf Boost 1.90 (Ubuntu 26.04), 1.91 (openSUSE Tumbleweed) und 1.92 (Arch) durchgängig verifiziert. Das belegt, dass der Server kompiliert und startet — nicht, dass er stundenlang einen Empfänger bedient. Behandeln Sie alle drei als im Produktivbetrieb ungetestet, bis jemand berichtet.

openSUSE heißt hier Tumbleweed. Dort ist `install_opensuse.sh` geprüft. Leap 15.6 funktioniert nicht: In seinen Repositories gibt es überhaupt kein `liquid-dsp-devel`, das das Backend braucht, und Boost liegt nur unter versionierten Paketnamen vor. Leap zu unterstützen hieße, fremde OBS-Repositories einzubinden — das bleibt vorerst außen vor.

Was das Installationsskript erledigt

Nichts muss vorher von Hand vorbereitet werden — keine Paketliste zum Kopieren, kein Node.js zum Nachladen, keine OpenCL-Pakete zum Suchen. Er läuft in 19 nummerierten Schritten und hält für bis zu zehn Fragen an, jede in einem „IHRE EINGABE WIRD BENÖTIGT“-Rahmen — bleiben Sie also entweder an der Tastatur, oder setzen Sie `PHANTOM_NONINTERACTIVE=1` und lassen Sie alles mit den Vorgaben beantworten (siehe unten). Lassen Sie ihn also nicht unbeaufsichtigt, und rechnen Sie je nach Maschine und Anzahl der Extras mit etwa zwanzig Minuten bis deutlich über einer Stunde.

#	Schritt	Was Sie gefragt werden
1	Listet die gerade laufenden PhantomSDR-Plus-Dienste auf — Admin-Panel, Reverse-Proxy, Statistikserver, Empfänger — und bietet an, sie zu stoppen, bevor irgendetwas angefasst wird. Nur ein laufendes Startskript gilt als Empfänger; eines, das nur in einem Editor geöffnet ist, bleibt unberührt.	bestätigen, Vorgabe ja
2–6	Erkennt die Distribution und installiert alle Build-Abhängigkeiten (Compiler, meson/ninja, FFTW, Boost, FLAC, Opus, liquid-dsp, zlib/zstd, libcurl ...) und dabei Node.js 22 über nvm, falls das System keines oder ein zu altes hat	nichts
7	Baut das Backend mit meson	nichts
8	Baut den Treiber für Ihren Empfänger — RX888 MkII / RX888, RTL-SDR (Blog V4 wird getrennt gefragt), SDRplay RSP1A, RigExpert Fobos SDR, Airspy HF+, HackRF One oder keinen. Beim RX888 werden auch die udev-Regeln installiert , damit der Server nie <code>sudo</code> für das Gerät braucht. RSP1A, Fobos und Airspy HF+ laufen über SoapysDR: ihre Wahl startet <code>setup-rsp1a.sh</code> , <code>setup-fobos.sh</code> oder <code>setup-airspyhf.sh</code> , die den Treiber und <code>rx_sdr</code> bauen und ebenfalls eine udev-Regel installieren (siehe Empfänger über SoapysDR (RSP1A, Fobos, Airspy HF+)) Der HackRF One braucht kein SoapysDR: seine Wahl startet <code>setup-hackrf.sh</code> , das das Paket <code>hackrf</code> der Distribution und eine udev-Regel installiert.	welchen SDR Sie haben
9	Öffnet <code>frontend/site_information.json</code> in Ihrem Editor	Rufzeichen, Locator, Hardware, Antenne — nicht überspringen
10–11	Installiert die Frontend-Abhängigkeiten und baut die Desktop- und /mobile-Seiten	nichts
12	Installiert OpenCL und wählt den Anbieter anhand der gefundenen Hardware (Intel-, AMD- oder NVIDIA-GPU bzw. die x86-CPU-Laufzeit). Gibt es kein OpenCL-fähiges Gerät, sagt es das und fährt fort	bestätigen, Vorgabe ja
13–15	Installiert das Admin-Panel , den FreeDV-RADE-V1-Decoder und den Statistikserver — alle drei standardmäßig	jeweils bestätigen, Vorgabe ja; jedes stellt eigene Fragen
16	Kopiert die fünf gepatchten websocketpp-Header erneut über das Meson-Subprojekt und prüft, dass sie angekommen sind. Drei davon sind die Boost- $\geq$ 1.87-Kompatibilitätsarbeit, ohne die das Backend unter Boost 1.90 nicht kompiliert; die anderen beiden sind projekteigene Änderungen, eine davon die Korrektur, die die Registrierung bei websdr.org braucht	nichts
17	Installiert die Emulation von KiwiSDR-Clients durch Ausführen von <code>kiwi_install.sh</code> , damit sich Kiwi-Clients wie AetherSDR mit diesem Empfänger verbinden können. Patcht die Quellen und ergänzt <code>[kiwi_emulation]</code> in den Konfigurationsdateien im Wurzelverzeichnis — siehe Emulation von KiwiSDR-Clients	bestätigen, Vorgabe ja
18	Führt <code>recompile.sh</code> aus, damit alles aus den gepatchten Quellen gebaut wird	[3] Both backend and frontend → Startvariante → [1] <code>build-all.sh</code>
19	Gibt die Zusammenfassung aus: jeder Schritt mit seinem Ergebnis und jede Komponente mit dem, was installiert wurde	nichts

Unbeaufsichtigt installieren

Jede Frage hat eine Umgebungsvariable, die sie überschreibt, und das Installationsskript wechselt von selbst auf die Vorgaben, wenn stdin kein Terminal ist (eine Pipe, ein Container, ein CI-Job). Das gilt in

allen vier gleichermaßen: `install.sh`, `install_arch.sh`, `install_fedora.sh` und `install_opensuse.sh`. Mit `PHANTOM_NONINTERACTIVE=1` läuft alles durch, ohne zu fragen:

Variable	Wirkung
<code>PHANTOM_NONINTERACTIVE=1</code>	answer every question with its default
<code>PHANTOM_SDR=1...7</code>	RX888 · RTL-SDR · SDRplay RSP1A · skip · Fobos · Airspy HF+ · HackRF (default 4)
<code>PHANTOM_RTLSDR_V4=y\ n</code>	RTL-SDR Blog V4 driver (default n)
<code>PHANTOM_SITE_EDIT=y\ n</code>	open <code>site_information.json</code> in an editor
<code>PHANTOM_OPENCL=y\ n</code>	install OpenCL (default y)
<code>PHANTOM_OPENCL_PROVIDER=1\ 2\ 3</code>	Intel · Mesa/Rusticl · POCL (default: from the detected hardware)
<code>PHANTOM_ADMIN=y\ n</code>	admin panel (y interactive, n unattended)
<code>PHANTOM_RADE=y\ n</code>	RADE / FreeDV (y interactive, n unattended)
<code>PHANTOM_STATS=y\ n</code>	statistics server (y interactive, n unattended)
<code>PHANTOM_KIWI=y\ n</code>	KiwiSDR client emulation (default y)
<code>PHANTOM_RECOMPILE=y\ n</code>	final rebuild (default y)
<code>PHANTOM_CURLPP=y\ n</code>	ohne curlpp fortfahren — nur Arch und openSUSE (Vorgabe y)
<code>PHANTOM_FIX_CLOCK_SKEW=y\ n</code>	Zeitstempel der Quelldateien zurücksetzen, die in der Zukunft liegen, damit meson bauen kann (Vorgabe y)

Die drei mit *n unbeaufsichtigt* markierten Unterinstallationsskripte sind selbst interaktiv, ein unbeaufsichtigter Lauf überspringt sie deshalb, statt an ihren Fragen hängen-zubleiben. Nennen Sie sie ausdrücklich, um sie einzuschließen:

```
# a complete unattended install for an RX888 MkII
PHANTOM_NONINTERACTIVE=1 PHANTOM_SDR=1 ./install.sh

# ... and with the admin panel, RADE and the statistics server too
PHANTOM_NONINTERACTIVE=1 PHANTOM_SDR=1 \
  PHANTOM_ADMIN=y PHANTOM_RADE=y PHANTOM_STATS=y ./install.sh
```

Nach einem Neustart sind die RX888-udev-Regeln und ein eventueller OpenCL-Treiber vollständig wirksam; das Skript sagt Ihnen, wann ein Neustart oder eine neue Anmeldung nötig ist.

Beim Einrichten des Admin-Panels bietet der Installer auch die beiden systemd-Units an und **empfiehlt dringend**, sie anzunehmen: Der CPU-Überhitzungsschutz läuft im Panel, ohne sie lässt ein Neustart oder ein Absturz die Maschine ungeschützt. Das Angebot entfällt automatisch dort, wo systemd nicht läuft (Container, WSL1, OpenRC) — siehe [Admin-Panel-Handbuch](#).

Jedes Extra ist ein normales Skript, das Sie jederzeit auch einzeln starten können — `./setup_admin.sh` ([Handbuch](#)), `./install_rade.sh` ([Handbuch](#)), `./install-stats-server.sh` ([Handbuch](#)), `./setup-rx888-udev.sh` und die Empfänger-Skripte `./setup-rsp1a.sh`, `./setup-fobos.sh`, `./setup-airspyhf.sh` und `./setup-hackrf.sh` — genau diese ruft das Installationsskript auf.

Von Hand bauen (Referenz)

WICHTIG

Für eine normale Installation brauchen Sie diesen Abschnitt nicht. `./install.sh` — bzw. das Installationsskript Ihrer Distribution — erledigt all das für Sie; siehe [Was das Installationsskript erledigt](#). Das Folgende ist eine Referenz für eine manuelle Einrichtung, für eine Distribution, die keines der Skripte abdeckt, oder um einen einzelnen Schritt von Hand zu reparieren.

Falls Sie einmal ohne das Installationsskript bauen müssen — eine Distribution, die keines der Skripte abdeckt, oder eine halb fertige Installation reparieren:

Backend bauen

```
# Clean any previous builds, Configure build with optimization, Compile
rm -rf build
meson setup build --buildtype=release --optimization=3
meson compile -C build

# Verify binary was created
ls -lh build/spectrumserver
```

Frontend bauen

```
cd frontend

# Install dependencies, Fix any vulnerabilities, Build the frontend
npm install
npm audit fix
npm run build

# Return to project root
cd ..
```

Installation überprüfen

```
# Check if binary exists
ls -lh build/spectrumserver

# Check if frontend was built
ls -lh frontend/dist/

# List important files
ls -lh *.sh *.toml
```

Opus-Audiocodec installieren

WICHTIG

Für eine normale Installation brauchen Sie diesen Abschnitt nicht. `./install.sh` — bzw. das Installationsskript Ihrer Distribution — erledigt all das für Sie; siehe [Was das Installationsskript erledigt](#). Das Folgende ist eine Referenz für eine manuelle Einrichtung, für eine Distribution, die keines der Skripte abdeckt, oder um einen einzelnen Schritt von Hand zu reparieren.

Opus bietet gegenüber FLAC bessere Tonqualität und geringere Latenz.

1. Systembibliothek libopus installieren

```
sudo apt-get update
sudo apt-get install -y libopus0 libopus-dev
```

2. Opus-Decoder für das Frontend installieren

```
cd PhantomSDR-Plus/frontend

# Install npm dependencies if not already done, Install Opus WASM decoder, Fix any vulnerabilities,
# Rebuild frontend
npm install
npm install @wasm-audio-decoders/opus-ml
npm audit fix
npm run build

# Return to project root
cd ..
```

3. Opus-Installation überprüfen

```
# Check if Opus system library is installed
pkg-config --modversion opus

# Check if Opus npm package is installed
cd frontend
npm list @wasm-audio-decoders/opus-ml
cd ..
```

Autorun-Spot-Reporter (FT8/FT4/WSPR)

PhantomSDR-Plus bringt einen optionalen **Autorun-Spot-Reporter** mit (im Verzeichnis `autorun/`). Er dekodiert FT8/FT4/WSPR serverseitig direkt vom Empfänger und lädt die Spots zu den Reporting-Netzwerken hoch:

- **FT8 / FT4 → PSK Reporter** (IPFIX/UDP)
- **WSPR → wsprnet.org** (HTTP)

Er wird vollständig über den Reiter **Admin-Panel → „Spot Reporting“** gesteuert, und **das Reporting ist standardmäßig AUS** — es wird nichts gesendet oder hochgeladen, bis Sie es dort aktivieren. Der Decoder läuft als eigener Node.js-Daemon und greift den Ton des Empfängers lokal ab, benötigt also keine zusätzliche HF-Hardware.

Voraussetzungen

Voraussetzung	Anmerkungen
Node.js 22+	Dieselbe Laufzeitumgebung, die der Frontend-Build ohnehin braucht — im Node.js-Schritt installiert.
npm-Pakete ws + cbor-x	Werden über <code>autorun/node_modules</code> aufgelöst, einen Symlink auf das <code>node_modules</code> des Frontends (beide Pakete sind in <code>frontend/package.json</code> deklariert).
util-linux (taskset)	Die Schaltfläche „Start“ im Admin-Bereich bindet den Daemon per <code>taskset</code> an die E-Cores. Praktisch auf jeder Distribution vorhanden; die Installer fügen es ausdrücklich hinzu.
Rufzeichen + Locator	Werden aus <code>frontend/site_information.json</code> gelesen (<code>siteSysop</code> / <code>siteGridSquare</code>), sofern nicht im Admin-Reiter überschrieben. Spots werden unter diesem Rufzeichen hochgeladen.

⚠ **Melden Sie nur, was Sie tatsächlich empfangen.** Spots werden unter Ihrem Rufzeichen in öffentliche Netze hochgeladen — aktivieren Sie nur Bänder/Betriebsarten, die Ihr Empfänger wirklich hört, und verwenden Sie Ihren korrekten Locator.

Automatische Installation

`install.sh` (und die distributionsspezifischen Varianten `install_arch.sh`, `install_fedora.sh`, `install_opensuse.sh`) erledigen alles für Sie: Sie installieren Node.js 22 und `util-linux`, führen das `npm install` des Frontends aus und legen anschließend den Symlink `autorun/node_modules` automatisch an. Weitere Schritte sind nicht nötig — die Funktion ist einsatzbereit, sobald `install.sh` fertig ist.

Manuelle Installation

HINWEIS

Das Installationsskript erledigt dies bereits für Sie: Es legt den Symlink an und installiert `util-linux`. Nutzen Sie die folgenden Schritte nur, um eine Installation von Hand zu reparieren.

Wenn Sie manuell installiert haben (Option B), legen Sie den Symlink nach dem `npm install` des Frontends selbst an, damit der Daemon seine Abhängigkeiten auflösen kann:

```
cd ~/PhantomSDR-Plus

# 1. Frontend deps must be installed first (ws + cbor-x live there)
cd frontend && npm install && cd ..

# 2. Link the autorun daemon to the frontend's node_modules
ln -sfn ../frontend/node_modules autorun/node_modules

# 3. Ensure taskset is available (Debian/Ubuntu shown; use your package manager)
sudo apt install -y util-linux
```

Hinweis: `autorun/node_modules` wird von git ignoriert, ein frisches `git clone` enthält es also nie — der Symlink muss nach jedem sauberen Checkout (neu) angelegt werden. Die Installer erledigen das für Sie; der obige Befehl gilt nur für manuelle Einrichtungen.

Überprüfen

```
# The symlink should point at ../frontend/node_modules
ls -l autorun/node_modules

# ws + cbor-x must resolve
ls -d frontend/node_modules/ws frontend/node_modules/cbor-x

# taskset must be on PATH
command -v taskset
```

Öffnen Sie danach das Admin-Panel, wechseln Sie auf den Reiter **Spot Reporting**, tragen Sie Ihre Identität ein, kreuzen Sie die zu dekodierenden Bänder/Betriebsarten an, aktivieren Sie die Ziele und drücken Sie **Start**. Ein Abzeichen „REPORTING“ erscheint im Hauptwasserfall, solange das Reporting aktiv ist. (PSK Reporter lädt gebündelt alle 5 Minuten hoch und wsprnet alle 2 Minuten, ein frisch gestarteter Daemon zeigt daher in den ersten Minuten „0 sent“ — das ist normal.)

Danach werden zwei verschiedene Zähler angezeigt, die leicht zu verwechseln sind. Die Kacheln **SPOTS UPLOADED PER DECODER** zählen nur den aktuellen Lauf, Stop/Start setzt sie also auf null zurück; die Zahl neben dem Kontrollkästchen jedes Bands bzw. jeder Betriebsart ist der **Gesamtwert seit Beginn**, gespeichert in `autorun-totals.json` und damit über Neustarts hinweg erhalten. Beides beschreibt das [Admin-Panel-Handbuch](#), ebenso die Seite **Graphen**, die CPU-Takt, CPU-Last, Temperatur und Nutzer online über 15 Minuten bis 24 Stunden darstellt.

Wenn Ihre Maschine heiß läuft, bietet das Panel außerdem einen [thermischen Schutz](#), der den Server bei gefährlicher CPU-Temperatur stoppt und wieder startet, sobald sie abgekühlt ist. Seine Schwellen leitet er aus dem kritischen Grenzwert Ihrer eigenen CPU ab, es gibt also nichts zu rechnen, und er funktioniert mit jeder Start-/Stopp-Methode. Er wird mit dem Panel installiert, startet aber im reinen Protokollmodus: Er zeichnet auf, was er *getan hätte*, und ändert nichts, bis Sie ihn in den Einstellungen aktivieren. Dauerdekodierung hält eine CPU rund um die Uhr beschäftigt, daher lohnt sich nach einer Woche Autorun ein Blick in den CRASH-Reiter, um zu sehen, wie nah Ihre Maschine tatsächlich kommt.

Der Serverport wird automatisch erkannt. Der Daemon greift direkt auf den `[server]` port des spectrumservers zu (Loopback + Token, unter Umgehung des Proxys). Er liest diesen Port aus der Konfigurationsdatei, mit der der laufende spectrumserver gestartet wurde, und funktioniert damit ohne Konfiguration auf jedem Port — die Zeile `[autorun] tap backend: ...` in `autorun.log` zeigt, was er ermittelt hat. Lief Ihr Server beim Start des Daemons nicht oder haben Sie eine ungewöhnliche Einrichtung, legen Sie ihn in `autorun.json` fest: `json "server": { "host": "127.0.0.1", "port": 9002 }` Ein falscher Port zeigt sich als `504 / tap closed 1006` in `autorun.log`, wobei decodes bei 0 stehen bleibt.

Emulation von KiwiSDR-Clients (optional)

Seit v4.1.0 kann PhantomSDR-Plus auch das **KiwiSDR-Protokoll** beantworten, sodass Software, die für einen KiwiSDR geschrieben wurde — **AetherSDR**, `kiwiclient` und die übrigen — sich direkt mit Ihrem Empfänger verbindet, auf demselben Host und Port, den Sie ohnehin veröffentlichen. Sie ist aus, bis `[kiwi_emulation] enabled = true` in der Konfiguration steht, mit der Ihr Empfänger läuft.

Das Installationsprogramm bietet sie als Schritt 17 an; `./kiwi_install.sh` spielt sie auf einem bereits installierten Baum ein.

Vollständige Dokumentation: [Emulation von KiwiSDR-Clients](#) — was die Brücke tut, wie man sie installiert, jeder `[kiwi_emulation]`-Schlüssel, das Verbinden eines Clients, die Lautstärke, das S-Meter, die Wasserfallrate und eine Symptomtabelle.

Konfiguration

1. Konfigurationsdatei auswählen

Wählen Sie die passende Konfigurationsdatei für Ihr SDR:

- `config-rtl.toml` – RTL-SDR-Sticks
- `config-rspla.toml` – SDRplay RSP1A
- `config-airspyhf.toml` – Airspy HF+ Discovery
- `config-fobos-hf.toml` – RigExpert Fobos SDR, HF1/HF2 Direktabtastung (0-25 MHz)
- `config-fobos.toml` – RigExpert Fobos SDR, RF-Zweig (25-6000 MHz)
- `config-rx888mk2.toml` – RX888 MK2
- `config-hackrf.toml` – HackRF One
- `config.example.hackrf.toml` – HackRF One

In diesem Beispiel verwenden wir den RTL-SDR.

2. Konfigurationsdatei bearbeiten

```
nano config-rtl.toml
```

Wichtige Einstellungen

```
[server]
port = 9002                # Web interface port (or everything else)
html_root = "frontend/dist/" # Frontend location
otherusers = 1             # Show other users (1=yes, 0=no)
threads = 2                # Number of server threads

[websdr]
register_online = true      # Register on sdr-list.xyz (true/false)
name = "Your WebSDR Name"  # Display name
antenna = "Your Antenna Type" # e.g., "Vertical", "Loop", "Dipole"
grid_locator = "AB12cd"    # Your Maidenhead grid square
hostname = "your.domain.com" # Your domain or IP address

[input]
sps = 2048000              # Sample rate (adjust for your coverage)
fft_size = 131072          # FFT size (higher = better resolution)
brightness_offset = -10    # Waterfall brightness adjustment
frequency = 145000000       # Base frequency in Hz (145 MHz for 2m)
signal = "iq"              # "iq" for complex, "real" for real sampling
fft_threads = 2            # FFT processing threads
accelerator = "opencl"     # "none", "cuda", or "opencl"
audio_sps = 12000          # Audio sample rate (keep at 12000)
audio_compression = "opus" # "flac" or "opus"
smeter_offset = -2         # S-meter calibration
waterfall_size = 1024      # Waterfall FFT size
waterfall_compression = "zstd" # Waterfall compression

[input.driver]
name = "stdin"             # Input driver
format = "u8"              # Sample format for RTL-SDR

[input.defaults]
frequency = 145500000       # Default tuning frequency
modulation = "FM"          # Default modulation mode
```

Richtwerte für die Abtastrate

Abdeckung	Abtastrate	FFT-Größe
2 MHz	2048000	131072
3.2 MHz	3200000	131072
10 MHz	10000000	1048576
30 MHz	30000000	2097152
60 MHz	60000000	4194304

3. Standortinformationen konfigurieren

```
nano frontend/site_information.json
```

Bearbeiten Sie die folgenden Felder:

```
{
  "siteSysop": "YourCallsign",
  "siteSysopEmailAddress": "your@email.com",
  "siteGridSquare": "AB12cd",
  "siteCity": "Your City, Country",
  "siteInformation": "https://github.com/sv1btl/PhantomSDR-Plus",
  "siteHardware": "Computer specifications",
  "siteSoftware": "PhantomSDR-Plus v5.0.0",
  "siteReceiver": "Your SDR model",
  "siteAntenna": "Antenna description",
  "siteNote": "Additional information",
  "siteIP": "http://your.domain.com:9002",
  "siteSDRBaseFrequency": 0,
  "siteSDRBandwidth": 2048000,
  "siteRegion": 1,
  "siteChatEnabled": true
}
```

IARU-Regionen:

- **1:** Europa, Afrika, Naher Osten, Nordasien
- **2:** Amerika (Nord, Mitte, Süd), Karibik
- **3:** Asien-Pazifik, Ozeanien

4. Frequenzmarker anpassen (optional)

```
nano markers.json
```

Fügen Sie Ihre bevorzugten Frequenzen, Relaisfunkstellen und Rundfunksender hinzu.

5. Startskript bearbeiten

```
nano start-rtl.sh
```

Das Startskript ist ein eigenständiger Starter mit Watchdog. Bearbeiten Sie nur den Block **RECEIVER CONFIGURATION** nahe dem Anfang, damit die Empfängerargumente zu Ihrer Einrichtung passen:

```
# == RECEIVER CONFIGURATION (the only receiver-specific part) ==
RX_LABEL="RTL-SDR"
RX_COMM="rtl_sdr"                # process name to monitor/kill
CONFIG="$PHANTOMDIR/config-rtl.toml" # your .toml
FIFO="$PHANTOMDIR/rtl.fifo"
RX_ARGS="{RX_ARGS:-f 145000000 -s 2048000 -}" # receiver args (edit these)
```

Parameter innerhalb von `RX_ARGS`:

- `-f 145000000`: Mittenfrequenz (145 MHz)
- `-s 2048000`: Abtastrate (2,048 MSPS)

`CONFIG` verweist auf Ihre `.toml`. Sie müssen sonst **nichts** im Skript ändern — die Logik für Start/Neustart/Watchdog/Protokollierung ist allgemeingültig. Sie können die Empfängerargumente auch beim Start überschreiben, ohne die Datei zu bearbeiten: `RX_ARGS="-f 145000000 -s 2048000 -" ./start-rtl.sh` (`RX888_ARGS` für `start-rx888mk2.sh`).

Gerätespezifische SDR-Einrichtung

RTL-SDR

RTL-SDR-Werkzeuge installieren

```
sudo apt install -y rtl-sdr
```

RTL-SDR testen

```
rtl_test
```

Drücken Sie Strg+C zum Beenden. Sie sollten Informationen zur Abtastrate sehen.

Konfiguration bearbeiten

```
nano config-rtl.toml
```

Übliche Einstellungen für den RTL-SDR:

```
sps = 2048000
frequency = 145000000
signal = "iq"
format = "u8"
```

Empfänger über SoapySDR (RSP1A, Fobos, Airspy HF+)

SDRplay RSP1A, RigExpert Fobos SDR und Airspy HF+ erreichen den Server auf dieselbe Weise: ein SoapySDR-Treiber für das Gerät und `rx_sdr` (aus `rx_tools`), das die Samples an `spectrumserver` weiterreicht. Ein Skript pro Empfänger installiert die ganze Kette:

Empfänger	Skript	Installer-Option	Startskript	Konfiguration
SDRplay RSP1A	<code>./setup-rsp1a.sh</code>	3	<code>start-rsp1a.sh</code>	<code>config-rsp1a.toml</code>
RigExpert Fobos SDR	<code>./setup-fobos.sh</code>	5	<code>start-fobos-hf.sh</code> (HF) / <code>start-fobos.sh</code> (RF)	<code>config-fobos-hf.toml</code> / <code>config-fobos.toml</code>
Airspy HF+	<code>./setup-airspyhf.sh</code>	6	<code>start-airspyhf.sh</code>	<code>config-airspyhf.toml</code>

Jedes dieser Skripte:

- installiert `git`, `cmake`, einen Compiler, `libusb` und **SoapySDR** aus den Paketen Ihrer Distribution — es erkennt `apt`, `dnf`, `pacman` und `zypper` und funktioniert damit auf Debian/Ubuntu, Fedora, Arch und openSUSE gleichermaßen;
- baut den Treiber des Empfängers aus den Quellen in `sdr_drivers/` (nicht von `git` erfasst);
- baut **rx_tools** für `rx_sdr`, sofern `rx_sdr` noch nicht installiert ist — eine Kopie dient allen Empfängern;
- installiert eine **udev-Regel** und nimmt Sie in die Gruppe `plugdev` auf, damit das Startskript nie `sudo` für das Gerät braucht. **Melden Sie sich einmal ab und wieder an**, damit die Gruppe wirksam wird;
- prüft zum Schluss, dass SoapySDR den Treiber wirklich auflistet, und bricht sonst mit einer Fehlermeldung ab.

Starten Sie sie als normaler Benutzer — sie rufen `sudo` selbst auf. Sie sind idempotent: ein erneuter Lauf aktualisiert die Treiberquellen mit `git pull` und baut neu. Das Installationsskript ruft genau diese Skripte auf; auf einer bereits installierten Station starten Sie einfach das für Ihren Empfänger. Der gemeinsame Teil steht in `setup-sdr-common.sh`, das nicht allein gestartet wird. Zwei optionale Variablen gelten für alle drei: `SDR_USER=<Name>` nimmt ein anderes Konto in `plugdev` auf, `SDR_SKIP_DEPS=1` überspringt die Paketinstallation.

Prüfen Sie bei angeschlossenem Gerät, dass der Empfänger gefunden wird:

```
SoapySDRUtil --find="driver=soapyMiri"    # RSP1A
SoapySDRUtil --find="driver=fobos"        # Fobos
SoapySDRUtil --find="driver=airspyhf"     # Airspy HF+
```

SDRplay RSP1A

`setup-rsp1a.sh` installiert den **Open-Source-Treiber**: `libmirisdr-5`, den per Reverse Engineering entstandenen Treiber für die Mirics-Chips `MSi2500/MSi001` im RSP1 und RSP1A, mit seinem SoapySDR-Modul `SoapyMiri` (`driver=soapyMiri`). Er braucht keinen Hintergrunddienst und kein root. Das Skript sperrt außerdem die Kernel-Treiber `msi2500` und `msi001` in `/etc/modprobe.d/blacklist-msi2500.conf`, weil sie den RSP1A als V4L2-Radio belegen, bevor `rx_sdr` ihn öffnen kann. War das Gerät angeschlossen, während sie geladen waren, stecken Sie es einmal neu ein (oder starten Sie neu).

Die geschlossene API von SDRplay (`driver=sdrplay`, mit `SoapySDRPlay` und dem Dienst `sdrplay`) funktioniert weiterhin — die mitgelieferte Datei `instructions-for-rsp1a` beschreibt diese Installation. `start-rsp1a.sh` wählt selbst zwischen beiden: Hat SoapySDR den Treiber `sdrplay`, nutzt es die API und startet vorher den Dienst `sdrplay` neu, sonst `libmirisdr-5`. Eine auf der API eingerichtete Station läuft also unverändert weiter. Um einen Treiber zu erzwingen, starten Sie mit `RX_DRIVER=miri` oder `RX_DRIVER=sdrplay`. Der gewählte Treiber steht in `logwebsdr.txt` als RSP1A driver: `miri` oder RSP1A driver: `sdrplay`.

RigExpert Fobos SDR

Der Fobos liefert **immer nur einen Zweig gleichzeitig**, deshalb hat er zwei Startskripte, die sich wie jedes `start-*.sh` die Watchdog-Sperre teilen — der Start des einen beendet das andere:

- **start-fobos-hf.sh** — die Eingänge HF1/HF2 in Direktabtastung, 0-25 MHz, auf Port 9003. In dieser Betriebsart gibt es keinen Lokaloszillator: der ADC digitalisiert ständig 0-25 MHz (`-f 0` ist Absicht, und die Warnung *Failed to set center freq* von `rx_sdr` ist zu erwarten). `rx_sdr` liefert CF32 mit 50 Msps; der kleine Wandler `cf32_to_real` (von `setup-fobos.sh` aus `cf32_to_real.c` gebaut) behält den I-Kanal und skaliert ihn auf `s16`, daher verwendet `config-fobos-hf.toml` `signal="real"` und `format="s16"`.
- **start-fobos.sh** — der RF-Zweig, 25-6000 MHz, IQ, auf Port 9002. Wählen Sie das Fenster vor dem ersten Start: `-f` (Mitte) und `-s` (Abtastrate) in der Zeile `RX_ARGS` müssen zu `frequency=` und `sps=` in `config-fobos.toml` passen. Die mitgelieferten Werte (97 MHz, 20 Msps) sind nur ein Ausgangspunkt, um zu prüfen, dass es läuft.

WICHTIG

`config-fobos-hf.toml` wird mit `accelerator="opencl"` ausgeliefert: ein 50-Msps-Strom durch eine FFT mit 1M Bins ist so schwer wie die 60 Msps des RX888, und auf der CPU gehen Samples verloren und die Maschine wird heiß. Auf einem Rechner ohne OpenCL-Gerät startet `spectrumserver` mit dieser Einstellung nicht — setzen Sie dort `accelerator="none"`.

`SoapyFobosSDR` braucht zum Bauen die Header von `libfobos` und `libfobos-sdr-agile`, daher baut `setup-fobos.sh` beide, auch für die Standard-Firmware. Die Fobos-Unterstützung stammt aus einem Skript, das eine andere Station geschrieben hat, nachdem sie einen Fobos mit PhantomSDR-Plus in Betrieb genommen hatte; die HF-Werte in `config-fobos-hf.toml` sind die, die dort funktioniert haben.

Airspy HF+

`setup-airspyhf.sh` installiert `libairspyhf` aus der Distribution, wo sie paketierte ist (Debian und Ubuntu, Fedora, openSUSE), und baut sie aus den Quellen, wo nicht (Arch hat sie nur im AUR); danach baut es `SoapyAirspyHF` (`driver=airspyhf`). `start-airspyhf.sh` liefert 912 ksps IQ in CS16, die `config-airspyhf.toml` als `format="s16"` liest. Die mitgelieferte Datei `instructions-for-airspy` beschreibt den manuellen Weg.

RX888 MK2

RX888-Werkzeuge installieren

```
# Install rx_tools
git clone https://github.com/rxseger/rx_tools.git
cd rx_tools
mkdir build && cd build
cmake ..
make -j4
sudo make install
sudo ldconfig

# Install RX888 firmware and support
# Follow manufacturer instructions
```

rx888_stream ohne sudo ausführen (udev-Regeln)

Standardmäßig ist das Cypress-FX3-USB-Gerät des RX-888 nur für root zugänglich, sodass `rx888_stream` `sudo` bräuchte. Führen Sie das mitgelieferte Hilfsskript einmal aus, um udev-Regeln zu installieren und sich der Gruppe `plugdev` hinzuzufügen:

```
./setup-rx888-udev.sh          # re-runs itself with sudo automatically
```

Es schreibt `/etc/udev/rules.d/99-rx888.rules` für alle drei Cypress-FX3-Produkt-IDs (04b4:00f1/00f3 Bootloader und 04b4:8613 mit geladener Firmware), lädt udev neu und legt einen stabilen Symlink `/dev/rx888` an. **Melden Sie sich ab und wieder an** (wegen der Gruppenänderung) und **stecken Sie das Gerät einmal neu an**; danach laufen `rx888_stream` — und der Starter `start-rx888mk2.sh` — ohne `sudo`. Das Skript ist idempotent und kann gefahrlos erneut ausgeführt werden. Für ein anderes Konto: `RX888_USER=<name> ./setup-rx888-udev.sh`.

Konfiguration bearbeiten

```
nano config-rx888mk2.toml
```

Übliche Einstellungen für den RX888:

```
sps = 60000000
frequency = 0
signal = "real"
format = "s16"
fft_size = 4194304
```

HackRF One

Für den HackRF wird nichts gebaut: `hackrf_transfer` aus dem Paket `hackrf` Ihrer Distribution liefert die Samples direkt an `spectrumserver`. `./setup-hackrf.sh` (Installer-Option 7) installiert dieses Paket mit `apt`, `dnf`, `pacman` oder `zypper` und ergänzt eine udev-Regel und die Gruppe `plugdev`, damit `start-hackrf.sh` ohne `sudo` läuft — melden Sie sich danach einmal ab und wieder an.

```
./setup-hackrf.sh    # installs the hackrf package and the udev rule
hackrf_info         # with the HackRF plugged in: must list the board
```

`start-hackrf.sh` liefert jeweils ein Fenster. Wählen Sie es vor dem ersten Start: `-f` (Mitte) und `-s` (Abtastrate) in seiner Zeile `RX_ARGS` müssen zu `frequency=` und `sps=` in `config-hackrf.toml` passen. `-l` und `-g` sind die Verstärkungen von LNA (0-40 dB, 8-dB-Schritte) und VGA (0-62 dB, 2-dB-Schritte), `-a 1` schaltet den HF-Verstärker ein. Die mitgelieferten Werte (98 MHz, 20 Msps) sind nur ein Ausgangspunkt, um zu prüfen, dass es läuft.

`hackrf_transfer` schreibt **vorzeichenbehaftetes** 8-Bit-IQ, daher braucht die Konfiguration `format="s8"`. `config.example.hackrf.toml` enthielt bis September 2026 `u8`; damit ist jedes Sample um 128 verschoben und der Wasserfall zeigt nur Rauschen — prüfen Sie ältere Kopien davon.

Mehrere Empfänger auf einem Rechner

Ein Rechner kann mehr als einen der oben genannten Empfänger gleichzeitig betreiben — etwa einen RX-888 für Kurzwellen und einen RTL-SDR für 2 m — jeden als eigenen Server, mit einer Empfängerauswahl auf der Seite. Jeder weitere Empfänger wird mit `INSTANCE=<name> ./start-<radio>.sh` gestartet und in `instances/<name>/` eingerichtet; `receivers.toml` führt sie für den Proxy und für `start-all.sh` auf. Alles Weitere steht in [Mehrere Empfänger](#). Am schnellsten fügen Sie einen Empfänger mit `./add-receiver.sh` hinzu, das die Installer am Ende anbieten: Es installiert den Treiber (`setup-rtlsdr.sh` für einen RTL-SDR, das passende `setup-*.sh` für die anderen) und schreibt den Instanzordner und `receivers.toml` für Sie.

Test und Überprüfung

1. Testlauf

```
# For RTL-SDR
./start-rtl.sh
```

Damit startet der Server **im Hintergrund** (er löst sich ab und kehrt sofort zurück) und beginnt, in `logwebsdr.txt` zu protokollieren.

2. Auf Fehler prüfen

Das Skript protokolliert seinen Fortschritt in `logwebsdr.txt` — beobachten Sie ihn live:

```
tail -f logwebsdr.txt
```

Achten Sie auf:

- `Starting the initialization script of the PhantomSDR Server`
- `<receiver> running (PID ...)`
- `Server started normally`
- `Server ... FAILED ...` oder `ERROR: receiver binary '...' not found` (korrigieren Sie die Empfängerargumente bzw. die `.toml` oder installieren Sie das Empfängerwerkzeug und führen Sie das Startskript erneut aus)

3. Weboberfläche aufrufen

Öffnen Sie im Browser:

```
http://localhost:9002
```

(Ersetzen Sie die Portnummer durch Ihren konfigurierten Port)

4. Funktion überprüfen

- Der Wasserfall sollte angezeigt werden
- Beim Klick auf Signale sollte Ton wiedergegeben werden
- Das S-Meter sollte auf Signale reagieren
- Die Nutzerzahl sollte „1“ anzeigen

5. Von einem anderen Gerät testen

Von einem anderen Rechner in Ihrem Netzwerk:

```
http://YOUR_SERVER_IP:9002
```

6. Ressourcenverbrauch prüfen

```
# Monitor CPU usage
htop

# Monitor GPU usage (if OpenCL/CUDA enabled)
nvidia-smi # For NVIDIA
radeontop  # For AMD

# Monitor network usage
iftop
```

7. Server stoppen

Der Server läuft im Hintergrund unter einem Watchdog, **Strg+C stoppt ihn also nicht** (und der Watchdog würde ihn ohnehin neu starten). Verwenden Sie das gemeinsame Stopp-Skript, das für jeden Empfänger funktioniert — es beendet zuerst den Watchdog, dann den Empfänger und `spectrumserver`:

```
./stop-websdr.sh
```

Automatischen Start einrichten

Mit systemd (empfohlen)

1. Service-Datei anlegen

```
sudo nano /etc/systemd/system/phantomsdr.service
```

Fügen Sie den folgenden Inhalt ein (Pfade und Benutzer anpassen):

```
[Unit]
Description=PhantomSDR-Plus WebSDR Server
After=network.target

[Service]
Type=simple
User=youruser
WorkingDirectory=/home/youruser/PhantomSDR-Plus
# Run the watchdog in the FOREGROUND (note the --watchdog flag) so systemd can
# track it. Do NOT use the plain "./start-rtl.sh" here – that form detaches into
# the background and exits, which systemd would treat as the service stopping.
ExecStart=/home/youruser/PhantomSDR-Plus/start-rtl.sh --watchdog
ExecStop=/home/youruser/PhantomSDR-Plus/stop-websdr.sh
Restart=always
RestartSec=10

[Install]
WantedBy=multi-user.target
```

Der `--watchdog`-Prozess startet Empfänger und spectrumserver bereits selbst neu; `Restart=always` ist nur eine Rückfallebene für den seltenen Fall, dass der Watchdog selbst beendet wird. Da systemd den Server hier überwacht, können Sie `systemctl start/stop/restart` und `journalctl -u phantomsdr -f` verwenden, statt die Skripte von Hand auszuführen.

2. Dienst aktivieren und starten

```
# Reload systemd
sudo systemctl daemon-reload

# Enable service to start on boot
sudo systemctl enable phantomsdr.service

# Start service now
sudo systemctl start phantomsdr.service

# Check status
sudo systemctl status phantomsdr.service
```

3. Dienst verwalten

```
# Start
sudo systemctl start phantomsdr

# Stop
sudo systemctl stop phantomsdr

# Restart
sudo systemctl restart phantomsdr

# View logs
sudo journalctl -u phantomsdr -f
```

Mit Screen (Alternative)

Meist unnötig: `./start-rtl.sh` löst sich bereits in den Hintergrund ab (per `setsid`) und läuft nach dem Abmelden mit eigenem Watchdog weiter. Screen ist nur dann praktisch, wenn Sie ausdrücklich eine interaktive Sitzung für die Vordergrundform `./start-rtl.sh --watchdog` möchten.

1. Screen installieren

```
sudo apt install -y screen
```

2. In einer Screen-Sitzung starten

```
screen -S phantomsdr
./start-rtl.sh
```

Drücken Sie Strg+A und dann D zum Ablösen.

3. Sitzung wieder aufnehmen

```
screen -r phantomsdr
```

CPU-Überhitzungsschutz

Vollständiges Handbuch: [THERMAL_GUARD.md](#) — die vier Modi und was ein Sysop bei jedem tun muss, Betrieb ohne Panel, die Throttle-Stufe ohne root, Testen und Fehlersuche.

PhantomSDR-Plus bringt einen Wächter mit, der den Server stoppt, wenn die CPU eine gefährliche Temperatur erreicht, und ihn wieder startet, sobald sie abgekühlt ist. Seine Schwellen leitet er aus dem kritischen Grenzwert ab, den Ihre eigene CPU veröffentlicht, es gibt also nichts zu rechnen, und er funktioniert mit jeder Start-/Stopp-Methode.

Wenn Sie das Admin-Panel betreiben, haben Sie ihn bereits — er läuft im Panel und wird auf dessen Seite „Einstellungen“ konfiguriert. Siehe das [Admin-Panel-Handbuch](#). Der Rest dieses Abschnitts gilt für Installationen **ohne** Panel.

1. Prüfen, was der Wächter auf Ihrer Maschine sieht

```
cd ~/PhantomSDR-Plus
python3 thermal_guard.py --once
```

```
sensor      : hwmon:coretemp
temperature: 58.0 C
crit        : 100.0 C
mode        : log
thresholds  : warn 88.0  throttle 92.0  stop 95.0  resume 75.0 (C)
cpufreq     : not writable – throttle stage disabled
```

Verwendet werden nur echte CPU-Die-Sensoren (`coretemp`, `k10temp`, `zenpower`, `cpu_thermal`); `acpitz` und unbeschriftete Thermal Zones werden bewusst ignoriert, weil sie oft eine Gehäuse- oder Platinentemperatur zig Grad unter der CPU melden. Steht dort `sensor : NONE`, lässt sich diese Maschine nicht schützen — häufig auf einem VPS oder in einem Container —, und der Wächter bleibt untätig, statt es vorzutäuschen.

Dafür muss nichts installiert werden: `thermal_guard.py` kommt mit der Python-Standardbibliothek aus.

2. Konfigurieren

Der Wächter liest `admin_config.json` neben `thermal_guard.py`. Legen Sie die Datei an, falls Sie keine haben — ohne Panel haben Sie keine. Das Nötigste:

```
{
  "sdr_process_name": "spectrumserver",
  "stop_script": "stop-websdr.sh",
  "start_script": "start-rx888mk2.sh",
  "thermal_mode": "stop+restart"
}
```

- `thermal_mode` — `log` (nur aufzeichnen, der Standard), `throttle`, `stop` oder `stop+restart`. **Diesen Wert müssen Sie setzen**, denn der Wächter wird untätig ausgeliefert: unverändert schreibt er nur auf, was er *getan hätte*. Sie können ihn auch auf der Kommandozeile als `--mode stop+restart` übergeben, was die Datei überschreibt — sind Sie mit allen anderen Standardwerten zufrieden, brauchen Sie also gar keine Konfigurationsdatei.
- `stop_script` / `start_script` — womit der Wächter Ihren Server stoppt und startet. Ohne Stopp-Skript greift er auf `SIGTERM` und nach 10 Sekunden auf `SIGKILL` für `sdr_process_name` zurück. Ohne Start-Skript stoppt er, startet aber nie neu.
- Schwellen und Zeiten lassen sich hier ebenfalls setzen, mit denselben Schlüsselnamen wie im Panel — die vollständige Tabelle steht im [Admin-Panel-Handbuch](#).

Wenn systemd Ihren SDR-Server überwacht, lassen Sie `stop_script` auf ein kleines Skript zeigen, das `systemctl stop ihre-unit` ausführt, statt den Wächter den Prozess direkt signalisieren zu lassen. Der Wächter gewinnt ohnehin — solange die CPU zu heiß ist, wiederholt er den Stopp alle 2 Sekunden, sodass alles, was den Server wiederbelebt, rückgängig gemacht wird —, aber ein sauberer Stopp ist besser als ein Kampf im Zwei-Sekunden-Takt.

3. Als Dienst betreiben

Das Repository enthält eine fertige Unit, `thermal-guard.service`. Passen Sie darin `User=` und die beiden Pfade an, dann:

```
sudo cp thermal-guard.service /etc/systemd/system/
sudo systemctl daemon-reload
sudo systemctl enable --now thermal-guard
systemctl status thermal-guard
```

`User=` muss ein Konto sein, das Ihre Start-/Stopp-Skripte ausführen darf — normalerweise derselbe Benutzer, der den SDR-Server betreibt. **Ein Wächter, der als Benutzer läuft, der den Server nicht stoppen kann, lässt Sie ungeschützt zurück und sieht dabei geschützt aus.**

Um den Wächter aus der Unit statt aus der Konfigurationsdatei scharfzuschalten, ergänzen Sie `ExecStart` um `--mode`:

```
ExecStart=/usr/bin/python3 -u /home/sysop/PhantomSDR-Plus/thermal_guard.py --mode stop+restart
```

Ohne `systemd`:

```
cd ~/PhantomSDR-Plus
nohup python3 -u thermal_guard.py --mode stop+restart >> thermal.log 2>&1 &
```

`python3 thermal_guard.py --help` listet alle Optionen auf, auch `--config` für eine Konfigurationsdatei an einem anderen Ort als neben dem Skript.

4. Beobachten, dann vertrauen

Alles, was der Wächter tut, landet in `crash.log` im PhantomSDR-Plus-Verzeichnis, ein Ereignis pro Zeile:

```
[THERMAL] warn: 89.0C (warn=88.0 crit=100.0) sustained 31s
[THERMAL] STOPPED server: 96.0C sustained 62s (stop=95.0 crit=100.0) – Started stop-websdr.sh
[THERMAL] lockout: process reappeared at 94.0C – re-stopped [3 time(s) so far]
[THERMAL] recovered: 74.0C held below 75.0 – lockout cleared
```

```
grep THERMAL crash.log
```

Lassen Sie `thermal_mode` zunächst eine Woche auf `"log"` und lesen Sie die Datei nach Ihren harten Momenten — einem vollen Rebuild, einem heißen Nachmittag. Steht dort nichts, kam Ihre Maschine nie in die Nähe. Stellen Sie dann den Modus auf `stop`.

Bevor Sie sich darauf verlassen, prüfen Sie den ganzen Weg einmal mit einer fiktiven Messung oberhalb Ihrer Stopp-Schwelle:

```
"thermal_test_temp": 97
```

Der Wächter behandelt sie als echt, sodass `warn` → `stop` → Sperre → Erholung → Neustart auf Wunsch ablaufen. Entfernen Sie die Zeile (oder setzen Sie sie auf `null`), um zum echten Sensor zurückzukehren. Im Modus `stop` wird der Server dabei wirklich gestoppt und Ihre Hörer werden getrennt — machen Sie das also, wenn niemand zuhört, oder testen Sie im Modus `log`, wo gezeigt wird, was passiert *wäre*, ohne etwas anzufassen.

Fehlerbehebung

Build-Fehler

Abhängigkeitsprobleme

```
# Ubuntu: Install missing dependencies
sudo apt install --fix-broken

# Fedora: Install missing dependencies
sudo dnf install <package-name>
```

Meson-Konfiguration schlägt fehl

```
# Clean and reconfigure
rm -rf build
meson setup build --buildtype=release
```

meson setup bricht mit ModuleNotFoundError: No module named 'mesonbuild' ab

```
Traceback (most recent call last):
  File "/home/<user>/local/bin/meson", line 3, in <module>
    from mesonbuild.mesonmain import main
ModuleNotFoundError: No module named 'mesonbuild'
```

Am Quellbaum ist nichts falsch. Eine übrig gebliebene Installation per `pip install --user meson` hat ein Startskript in `~/local/bin` hinterlassen, das im `PATH` vor `/usr/bin` steht und damit die funktionierende Kopie der Paketverwaltung verdeckt. Ein Distributions-Upgrade (etwa Ubuntu 24.04 → 26.04) stellt Python auf eine neue Version um, das alte `site-packages` mit `mesonbuild` liegt nicht mehr im Importpfad, und das Skript stirbt, bevor es irgendetwas tut. Dasselbe kann `ninja` passieren.

Das Installationsprogramm erkennt das, umgeht es für die Dauer des Laufs und warnt Sie dabei — reparieren Sie das System dennoch:

```
rm -f ~/.local/bin/meson
hash -r
meson --version          # muss eine Version ausgeben
```

Wenn Sie meson lieber weiterhin über pip installiert lassen möchten, installieren Sie es für das Python neu, das dieses System jetzt hat:

```
python3 -m pip install --user --force-reinstall --break-system-packages meson
```

meson setup bricht mit Clock skew detected ab

```
ERROR: Clock skew detected. File /home/pi/PhantomSDR-Plus/build/./meson.build has a time stamp
10392.2144s in the future.
```

Mit dem Quellbaum ist nichts verkehrt: die Systemuhr geht den Dateien hinterher. meson und ninja bauen nicht, wenn eine Eingabe neuer als die aktuelle Zeit ist, denn sie können nicht erkennen, was veraltet ist. Das passiert auf einem Raspberry Pi ohne Zelle in der RTC-Halterung — jeder Start beginnt bei der zuletzt bekannten Zeit, und ein Build vor der NTP-Synchronisierung sieht den ganzen Baum in der Zukunft datiert — und bei jedem Baum, der von einer vorgehenden Maschine entpackt oder kopiert wurde.

Stellen Sie zuerst die Uhr richtig:

```
timedatectl                      # stimmt die Zeit? ist NTP synchronisiert?
sudo timedatectl set-ntp true
```

Warten Sie ein paar Sekunden und starten Sie das Installationsprogramm erneut. Es prüft das vor dem Aufruf von meson und bietet an, die betroffenen Zeitstempel zurückzusetzen (PHANTOM_FIX_CLOCK_SKEW=yn). Von Hand, im Quellverzeichnis:

```
find . -path ./build -prune -o -path ./git -prune -o -newermt now -print0 | xargs -0r touch
```

Das Backend wurde gebaut, aber es gibt keine Webseite

Das Symptom ist eine Installation, die weitgehend geglückt aussieht: build/spectrumserver ist da, der Server startet, und der Browser bekommt nichts — weil frontend/dist/ nie erzeugt wurde. Suchen Sie in der Ausgabe des Installationsskripts nach:

```
[vite-plugin-top-level-await] missing field `type`
  at Compiler.printSync (node_modules/@swc/core/index.js:257:29)
error during build:
  Failed: Default
```

vite-plugin-top-level-await verlangt @swc/core ^1.12.14, daher löst ein frisches npm install auf 1.16.0 auf, dessen printSync() den Syntaxbaum ablehnt, den das Plugin ihm übergibt. Mit Ihrer Distribution hat das nichts zu tun — jedes Installationsskript pinnt jetzt die funktionierende Version, und frontend/package.json ebenfalls. Eine vor diesem Pin eingerichtete Maschine läuft weiter, bis ihr node_modules gelöscht wird; deshalb taucht das bei einer Neuinstallation scheinbar aus dem Nichts auf.

Wenn Sie einen älteren Checkout von Hand reparieren:

```
cd frontend
npm pkg set 'overrides.@swc/core=1.15.33'
npm install
./build-all.sh
```

Laufzeitfehler

Port bereits belegt

```
# Find process using port
sudo lsof -i :9002

# Kill process
sudo kill -9 <PID>
```

Zugriff auf das SDR verweigert

```
# Add user to plugdev group
sudo usermod -a -G plugdev $USER

# Reload groups (or log out and back in)
newgrp plugdev
```

Bei einem **RX-888 MkII** reicht das allein nicht — das Cypress-FX3-Gerät benötigt zusätzlich udev-Regeln. Führen Sie das mitgelieferte Hilfsskript aus (es installiert die Regeln *und* fügt Sie zu `plugdev` hinzu), melden Sie sich dann ab und wieder an und stecken Sie das Gerät neu an:

```
./setup-rx888-udev.sh
```

Audioprobleme

```
# Try different codec
# Edit config file and change:
audio_compression = "flac" # or "opus"

# Rebuild
cd frontend && npm run build && cd ..
```

OpenCL-Probleme

clinfo zeigt keine Geräte

```
# Verify drivers are loaded
sudo clinfo

# Check OpenCL ICD files
ls /etc/OpenCL/vendors/

# Reinstall drivers (Intel example)
sudo apt remove --purge intel-opencl-icd
sudo apt install intel-opencl-icd
```

Keine Leistungsverbesserung

```
# Verify OpenCL is enabled in config
accelerator = "opencl"

# Try different device
# Add to config file:
[input.opencl]
device_id = 0 # Try 0, 1, 2, etc.
```

Netzwerkprobleme

Kein Zugriff von anderen Geräten

```
# Check firewall
sudo ufw status
sudo ufw allow 9002/tcp

# Or disable firewall temporarily for testing
sudo ufw disable
```

Hohe Latenz

```
# Reduce waterfall size in config
waterfall_size = 512

# Increase buffer size
buffer_size = 16384

# Use Opus instead of FLAC
audio_compression = "opus"
```

Probleme mit dem SDR-Gerät

RTL-SDR nicht gefunden

```
# Check if device is recognized
lsusb | grep Realtek

# Test with rtl_test
rtl_test

# Try blacklisting DVB-T drivers
echo "blacklist dvb_usb_rtl28xxu" | sudo tee /etc/modprobe.d/blacklist-rtl.conf
sudo rmmod dvb_usb_rtl28xxu
```

SDRplay nicht gefunden

`grep "RSP1A driver" logwebsdr.txt` zeigt, welchen Treiber `start-rsp1a.sh` gewählt hat.

Mit dem offenen Treiber (`libmirisdr-5`, `RSP1A driver: miri`):

```
# The kernel's msi2500 driver must not hold the device
lsmod | grep msi2500 && sudo modprobe -r msi2500 msi001

# Test with SoapySDR
SoapySDRUtil --find="driver=soapyMiri"
```

Mit der API von SDRplay (`RSP1A driver: sdrplay`):

```
# Check API installation
systemctl status sdrplay

# Restart API
sudo systemctl restart sdrplay

# Test with SoapySDR
SoapySDRUtil --find="driver=sdrplay"
```

Fobos oder Airspy HF+ nicht gefunden

```
SoapySDRUtil --info | grep -i factories # must list fobos / airspyhf
SoapySDRUtil --find="driver=fobos"      # or driver=airspyhf
id | grep plugdev                       # log out and back in if missing
```

Fehlt der Treiber in der Liste, starten Sie `./setup-fobos.sh` oder `./setup-airspyhf.sh` erneut: es bricht mit dem Grund ab, wenn der Bau oder die SoapySDR-Prüfung scheitert.

Leistungsoptimierung

CPU-Optimierung

```
# Adjust thread counts based on CPU cores
[server]
threads = 4 # Set to number of cores

[input]
fft_threads = 4 # Set to number of cores
```

Speicheroptimierung

```
# Reduce memory usage
[input]
waterfall_size = 512 # Lower value = less memory
fft_size = 65536     # Lower value = less memory
```

Netzwerkoptimierung

```
# Optimize for limited bandwidth
[input]
audio_compression = "opus" # More efficient than FLAC
waterfall_compression = "zstd" # Efficient compression
```

PhantomSDR-Plus aktualisieren

Seit Version 4.1.0 enthält das Repository `update.sh` — ein Update-Werkzeug, das eine installierte Station auf den veröffentlichten Stand bringt, **ohne die Dateien anzurühren, die sie zu Ihrer Station machen**. Es ersetzt das handgeschriebene `git pull`-Skript, das frühere Ausgaben dieser Anleitung anzulegen empfahlen, und benötigt `git` überhaupt nicht: der veröffentlichte Baum wird als Tarball geladen und Datei für Datei mit Ihrem verglichen. Es funktioniert also gleich, ob Sie das Repository geklont, ein `update.zip` entpackt oder den Baum von einem USB-Stick kopiert haben.

Wenn Ihre Installation update.sh noch nicht hat

Ein älterer Baum enthält das Skript nicht. Holen Sie es einmal — es ist der einzige Schritt dieses ganzen Verfahrens, den Sie je von Hand tun:

```
cd ~/PhantomSDR-Plus
curl -fLO https://raw.githubusercontent.com/sv1btl/PhantomSDR-Plus/main/update.sh
chmod +x update.sh
```

Von da an wird alles — Quellen, Frontend, Dokumentation, Installer und update.sh selbst — vom Werkzeug geholt.

Schritt 1 — ansehen, was sich ändern würde (es wird nichts geschrieben)

```
cd ~/PhantomSDR-Plus
./update.sh
```

Der veröffentlichte Baum wird geladen, mit Ihrem verglichen und ein Bericht ausgegeben. Geschrieben wird dabei nichts, der Aufruf ist also jederzeit gefahrlos — auch auf einer Station, die gerade auf Sendung ist. Der Rückgabewert ist 0, wenn alles aktuell ist, und 10, wenn ein Update bereitliegt; so kann ein Cron-Job Sie benachrichtigen, wenn es etwas zu tun gibt.

Schritt 2 — anwenden

```
./update.sh --apply
```

Drei Arten von Dateien werden unterschiedlich behandelt, und genau darauf kommt es an:

Dateien	Was geschieht
config.toml, config-<empfänger>.toml, markers.json, admin_config.json, autorun.json, frequencylist/, chat_history.txt, frontend/variant.json, frontend/site_information.json, die Logdateien, build/, frontend/dist/	Werden nie angerührt und tauchen in keiner Rückfrage auf. Sie sind es, die aus dem Rechner <i>Ihre</i> Station machen.
start-*.sh, stop-websdr.sh, die *.service-Units, install*.sh, recompile.sh, setup_admin.sh, smeter_theme.sh, proxy.py, admin_server.py, thermal_guard.py, websdr_relay.py, frontend/src/bands-config.js, die Beispielkonfigurationen config.example.*.toml	Es wird immer nachgefragt , denn das sind die Dateien, die ein Betreiber aus gutem Grund geändert haben kann.
Alles Übrige	Wird aktualisiert, nachdem eine Kopie der alten Datei im Sicherheitsarchiv gespeichert wurde (siehe <i>Ein Update rückgängig machen</i>).

Für jede Datei der mittleren Gruppe werden die Unterschiede angezeigt und drei Möglichkeiten angeboten:

```
start-rx888mk2.sh [K]eep mine / [u]pstream / [b]oth (ENTER = Keep mine)
```

- **Keep mine** — Ihre Datei bleibt genau so, wie sie ist.
- **upstream** — die neue Fassung wird eingespielt, Ihre wird vorher gesichert.
- **both** — die neue Fassung wird als start-rx888mk2.sh.new daneben abgelegt, damit Sie Ihre eigenen Änderungen in Ruhe übernehmen können.

Wie sich der erste Lauf anfühlt. Beim ersten Mal gibt es keine Aufzeichnung darüber, aus welcher Fassung Ihre Dateien stammen, also wird jede Datei der mittleren Gruppe Ihnen vorgelegt — etwa zehn Fragen. Antworten Sie so:

Ihre Lage	Antwort
Sie haben die Datei nie geändert	u — neue Fassung übernehmen. Der Normalfall.
Sie haben sie geändert (eigene RX888_ARGS, CPU-Pinning, angepasste Unit)	b — Ihre bleibt, die neue landet als <Datei>.new daneben.
Sie sind unsicher	ENTER — Ihre bleibt, nichts geht verloren, Sie können später vergleichen.

Ihre Konfiguration ist davon nie betroffen: gefragt wird nur nach Skripten und Service-Units.

update.sh merkt sich in .update-state/ die Fassung jeder eingespielten Datei. Ab dem zweiten Lauf kann es daher eine Datei, die **Sie** geändert haben, von einer bloß alten unterscheiden und fragt nur noch bei denen nach, die Sie wirklich angefasst haben.

Bevor irgendetwas geschrieben wird, hält es Empfänger, Admin-Panel, Reverse-Proxy und das WebSDR-Diversity-Relay **der Installation an, die es aktualisiert** — was zu einem anderen Verzeichnis gehört, wird aufgeführt und weiterlaufen gelassen, sodass ein zweiter Klon aktualisiert werden kann, während der erste auf Sendung bleibt — und startet am Ende genau das wieder, was es angehalten hat. Haben sich Quell- oder Frontend-Dateien geändert, bietet es an, recompile.sh für Sie auszuführen. Gelöscht wird nie etwas: Dateien, die aus dem Repository verschwunden sind, werden gemeldet und nur auf ausdrückliche Anforderung mit --prune entfernt.

Ein Update rückgängig machen

```
./update.sh --restore LAST
```

Jede Datei, die ein Update überschreibt, wird vorher gesichert, in einem datierten Archiv pro Lauf im normalen, sichtbaren Ordner update-backups/ :

```
update-backups/phantomsdr-backup-20260923-164530.zip
```

Das Archiv enthält nur die Dateien, die dieser Lauf ersetzt hat, jede unter dem Pfad, den sie in Ihrem Baum hat. Es lässt sich also als einzelne Datei auf einen anderen Rechner kopieren, überall öffnen oder direkt über die Installation zurückentpacken. Ist zip nicht installiert, wird es stattdessen als .tar.gz geschrieben. Die drei neuesten werden behalten. Jeder dieser Befehle stellt wieder her, was ein Update ersetzt hat:

```
./update.sh --restore LAST
./update.sh --restore phantomsdr-backup-20260923-164530.zip
```

Jedes Archiv enthält außerdem ein eigenes restore.sh, für den Fall, dass update.sh selbst das Problem ist: Archiv entpacken und ihm Ihr Installationsverzeichnis übergeben, zum Beispiel ./restore.sh /home/pi/PhantomSDR-Plus. Während der Wiederherstellung läuft update.sh aus einer Kopie außerhalb des Baums, weil es meist selbst zu den zurückgespielten Dateien gehört.

Sicherungen von vor dem 23. September 2026 liegen im versteckten Ordner .update-backups/<Zeitstempel>; sie lassen sich genauso wiederherstellen und zählen zu den drei behaltenen.

Weitere Optionen

```
./update.sh --apply --yes      # fragt nie; jede von Ihnen geänderte Datei BLEIBT
./update.sh --ref v5.0.0      # ein Tag, Branch oder Commit statt des aktuellen Standes
./update.sh --list-excludes    # zeigt die Nie-anrühren-Regeln, wie sie hier gelten
./update.sh --verbose         # listet jede Datei, nicht nur die ersten 40
```

Eigene Nie-anrühren-Regeln tragen Sie zeilenweise als Glob-Muster in `update-exclude.txt` im Wurzelverzeichnis der Installation ein.

Wenn der Neubau auf einer sehr alten Installation scheitert

`update.sh` aktualisiert Dateien, keine Systempakete. Ist Ihr Baum so alt, dass der Bau inzwischen Bibliotheken braucht, die Sie nicht haben, bricht `recompile.sh` mit einem Compiler- oder meson-Fehler ab. Das ist kein kaputtes Update — es fehlen die Abhängigkeiten:

```
./install.sh
```

Der Installer wird vom selben Lauf mit aktualisiert, und Ihre Konfiguration übersteht auch ihn.

Von Hand aktualisieren

Wenn Sie ein Dateipaket lieber selbst einspielen:

```
cd ~
unzip -o update.zip
cd PhantomSDR-Plus
./stop-websdr.sh
cp -a ~/update/. .
chmod +x *.sh
./recompile.sh
./start-rx88mk2.sh      # oder das Startskript Ihres Empfängers
```

Sichern Sie vorher Ihre Konfiguration — `config-*.toml`, `frontend/site_information.json`, `admin_config.json` und `markers.json` —, denn ein Dateipaket kann Ihre Änderungen nicht von denen der Veröffentlichung unterscheiden. Genau dieses Problem löst `update.sh`.

Sichern und Wiederherstellen

Zu sichernde Dateien

- Konfigurationsdateien: `*.toml`
- Standortinformationen: `frontend/site_information.json`
- Marker: `markers.json`
- Eigene Skripte: `start-*.sh`, `stop-*.sh`
- Chatverlauf: `chat_history.txt`
- Hintergrundbild: `frontend/src/assets/background.jpg`

Sicherungsbefehl

```
cd ~/PhantomSDR-Plus
tar -czf phantomsdr-backup-$(date +%Y%m%d).tar.gz \
  *.toml \
  *.sh \
  markers.json \
  chat_history.txt \
  frontend/site_information.json \
  frontend/src/assets/background.jpg
```

Wiederherstellungsbefehl

```
tar -xzf phantomsdr-backup-YYYYMMDD.tar.gz
```

Sicherheitsüberlegungen

Firewall-Konfiguration

```
# Allow only necessary ports
sudo ufw allow 9002/tcp
sudo ufw enable
```

Reverse-Proxy (optional)

Erwägen Sie nginx oder Apache als Reverse-Proxy für:

- SSL-/TLS-Verschlüsselung
- Zuordnung eines Domainnamens
- Lastverteilung
- Zugriffssteuerung

Nutzerbegrenzung

Bearbeiten Sie `config.toml`:

```
[server]
max_users = 100 # Limit concurrent users
```

Hilfe erhalten

Ressourcen

- **Dokumentation:** diese Anleitung, README.md, USER_GUIDE.md
- **GitHub Issues:** <https://github.com/sv1btl/PhantomSDR-Plus/issues>
- **Live-Demo:** <http://phantomsdr.no-ip.org:8900/>

Probleme melden

Geben Sie beim Melden von Problemen an:

1. Betriebssystem und Version

2. Modell des SDR-Geräts
3. Inhalt der Konfigurationsdatei
4. Fehlermeldungen
5. Systemressourcenverbrauch (CPU, RAM, GPU)

Unterstützung durch die Gemeinschaft

- Prüfen Sie vorhandene GitHub-Issues, bevor Sie neue anlegen
- Machen Sie ausführliche Angaben zu Ihrer Einrichtung
- Fügen Sie Protokolle und Fehlermeldungen bei
- Bleiben Sie geduldig und respektvoll

Anhang A: vollständige Abhängigkeitsliste

Paketliste für Ubuntu 24.04

```
build-essential
cmake
pkg-config
meson
libfftw3-dev
libwebsocketpp-dev
libflac++-dev
zlib1g-dev
libzstd-dev
libboost-all-dev
libopus-dev
libliquid-dev
git
util-linux (taskset – for the autorun spot reporter)
psmisc
wget
curl
rtl-sdr (for RTL-SDR)
airspy (for Airspy)
hackrf (for HackRF)
libclfft-dev (for OpenCL)
ocl-icd-opencl-dev (for OpenCL)
clinfo (for OpenCL)
```

Anhang B: Konfigurationsbeispiele

Beispiel 1: RTL-SDR für VHF/UHF

```
[input]
sps = 2048000
frequency = 145000000
signal = "iq"

[input.driver]
format = "u8"

[input.defaults]
frequency = 145500000
modulation = "FM"
```

Beispiel 2: RX-888 mk2 für Kurzwelle (0-30 MHz)

Der Empfänger, den die meisten Sysops einsetzen. Dies ist ein vollständiger [input]-Abschnitt und kein Ausschnitt; die Werte stammen aus der mitgelieferten `config-rx888mk2.toml`.

```
[input]
sps = 60000000          # 0-30 MHz per Direktabtastung
fft_size = 4194304      # siehe Hinweis unten
fft_threads = 8
brightness_offset = -9  # negativer, wenn der Wasserfall schwarze Flächen zeigt
frequency = 0           # Basisband: der RX-888 tastet ab DC ab
signal = "real"         # nicht "iq" - Direktabtastung liefert einen reellen Strom
accelerator = "opencl"  # "none", wenn keine OpenCL-Laufzeit vorhanden ist
audio_sps = 12000
audio_compression = "flac"
waterfall_size = 1024
waterfall_compression = "zstd"
smeter_offset = 5
analog_smeter_offset = 5

[input.driver]
name = "stdin"
format = "s16"

[input.defaults]
frequency = 7120000
modulation = "LSB"
```

Gespeist wird es von `rx888_stream`, was `start-rx888mk2.sh` für Sie übernimmt:

```
rx888_stream -f /path/to/SDDC_FX3.img -s 60000000 -g 50 -a 0 -m low --pga -d -r -o - \
| build/spectrumserver --config config-rx888mk2.toml
```

Zu `fft_size`: Bei 60 MSPS ist 4194304 die richtige Größe. 8388608 verdoppelt die Wasserfallauflösung und zugleich Speicher- und CPU-Bedarf jeder Transformation — auf den meisten Rechnern erkaufte man schärfere Bins mit verlorenen Frames. Beginnen Sie mit 4194304 und gehen Sie nur höher, wenn der Server deutlich Luft hat.

Beispiel 3: HackRF für Breitband-FM

```
[input]
sps = 10000000
frequency = 100900000
signal = "iq"

[input.driver]
format = "s8"

[input.defaults]
frequency = 100900000
modulation = "WBFM"
```

Installation abgeschlossen! Sie sollten nun einen voll funktionsfähigen PhantomSDR-Plus-Server haben.

73 de SV1BTL & SV2AMK

KAPITEL 3

Admin-Panel einrichten

ADMIN_PANEL_SETUP.md

PhantomSDR-Plus Admin-Panel — Einrichtungsanleitung

⚠ **Sicherheitshinweis:** Halten Sie dieses Admin-Panel in Ihrem Heimnetz oder hinter einem VPN. Eine öffentliche Freigabe des Admin-Ports ist ohne geeignete Absicherung der Authentifizierung nicht zu empfehlen.

Überblick

Das Admin-Panel besteht aus zwei Python-Diensten:

Dienst	Datei	Rolle
Admin-Panel	admin_server.py	Flask-Webanwendung, bindet an 127.0.0.1 (nur intern)
Reverse-Proxy	proxy.py	Stellt sowohl das SDR als auch das Admin-Panel auf einem einzigen öffentlichen Port bereit

Beide Dienste lesen ihre Konfiguration aus `admin_config.json`, das von `setup_admin.sh` geschrieben wird.

Ändern Sie Portkonstanten nicht direkt in den Python-Dateien — alle Ports und die IP des SDR-Hosts liegen in `admin_config.json`.

Was das Admin-Panel bietet

- **Dashboard** — Serverstatus in Echtzeit, CPU/RAM, Top-Prozesse, aktuelle Protokollausgabe, Terminal
- **Konfigurationseditor** — beliebige `.toml`-, `.sh`-, `.json`-, `.h`- und `.cpp`-Dateien Ihrer Installation ansehen, bearbeiten und speichern
- **Protokollanzeige** — jede Protokolldatei in Echtzeit verfolgen (SDR-Server, Admin-Panel, RADE, Absturz, Autorun, Proxy)
- **Marker** — Frequenzmarker ansehen und bearbeiten
- **Chatverlauf** — das WebSDR-Chatprotokoll ansehen, vollständig leeren oder einzelne Nachrichten löschen, ohne den Server neu zu starten
- **Wasserfallnachricht** — ein dauerhaftes Banner an alle verbundenen Nutzer senden, in Echtzeit auf der Wasserfallanzeige sichtbar
- **Nutzer** — Live-Liste der verbundenen Hörer mit ihrer Frequenz, Betriebsart, Dauer und einer -Kick-Schaltfläche
- **Löschen von Chatnachrichten** — -Delete-Schaltfläche, die genau diese eine Nachricht sofort aus dem Protokoll entfernt
- **Wasserfall-Rundnachrichten** — erlaubt es, ein dauerhaftes Textbanner in den Wasserfall jedes verbundenen Nutzers zu schieben
- **Spot Reporting** — den Autorun-Decoder für FT8/FT4/JS8/WSPR starten/stoppen, Bänder/Betriebsarten wählen und Spots an PSK Reporter / wsprnet melden (standardmäßig AUS), mit zwei

Zählern: Kacheln je Decoder für den aktuellen Lauf und ein Gesamtwert seit Beginn neben jedem Band-/Betriebsart-Kontrollkästchen

- **Graphen** — CPU-Frequenz, CPU-Last, CPU-Temperatur und verbundene Nutzer, aufgezeichnet über die letzten 15 Minuten / 1 Stunde / 4 Stunden / 12 Stunden / 24 Stunden
- **Thermischer Schutz (Thermal Guard)** — stoppt den Server, wenn die CPU überhitzt, und startet ihn wieder, sobald sie abgekühlt ist; die Schwellen leiten sich aus Ihrer eigenen CPU ab. Funktioniert mit jeder Start-/Stopp-Methode und läuft ausgeliefert nur im Protokollmodus, tut also nichts, bis Sie ihn aktivieren
- **Einstellungen** — Admin-Passwort, SDR-Basisverzeichnis, Prozessname und öffentlichen Port ändern

Voraussetzungen

- PhantomSDR-Plus bereits installiert und in Betrieb
- Python 3.8+

HINWEIS

Normalerweise erledigt das Installationsskript dies für Sie. `./install.sh` — ebenso jedes der vier Distributionsskripte, beide RADE-Skripte und das Statistikserver-Skript — richtet das Admin-Panel standardmäßig ein und installiert zuvor `pip` mit dem Paketmanager des Systems, falls es fehlt. Diese Seite beschreibt, was in diesem Schritt geschieht und was zu tun ist, wenn Sie ihn übersprungen haben oder später etwas ändern wollen.

Die Python-Bibliotheken des Panels (`flask`, `psutil`, `aiohttp`, `tomli-w`) werden von `setup_admin.sh` installiert; sollte das fehlschlagen, gibt es den genauen Befehl aus.

Schritt 1 — Dateien

Legen Sie diese vier Dateien in Ihr PhantomSDR-Plus-Verzeichnis:

```
admin_server.py
manage_admin.sh
setup_admin.sh
proxy.py
```

Machen Sie die Shell-Skripte ausführbar:

```
chmod +x setup_admin.sh manage_admin.sh
```

Schritt 2 — Das Einrichtungsskript ausführen

```
./setup_admin.sh
```

Das Skript wird:

1. prüfen, ob Python 3 installiert ist
2. das Vorhandensein von `admin_server.py` und `manage_admin.sh` überprüfen
3. `127.0.0.1` als `sdr_host` in der Konfiguration eintragen (siehe [Die Einstellung sdr_host](#))

4. nach drei Portnummern fragen. Jede Abfrage bietet einen Vorgabewert in Klammern, den ein einfaches Enter übernimmt, sodass eine normale Einrichtung drei Tastendrucke braucht:
5. **Spectrumserver-Port** — der Port, auf dem Ihr SDR-Server lauscht (Vorgabe 8900, genau das, was jede `config-*.toml` im Repository mitbringt)
6. **Interner Port des Admin-Panels** — wo `admin_server.py` lokal bindet (Vorgabe 3000)
7. **Öffentlicher Proxy-Port** — der einzige externe Port, der SDR + Admin vereint (Vorgabe 8902)

Ungültige Antworten werden abgelehnt und erneut abgefragt, aber nur fünfmal — danach wird der Vorgabewert verwendet. Ein Lauf, dessen Eingabe kein Terminal ist (Pipe, Cron, unbeaufsichtigt), übernimmt die Vorgaben sofort, statt auf eine Eingabe zu warten, die nie kommt.

1. `flask`, `psutil`, `aiohttp` und `tomli-w` per `pip` installieren
2. `ss` die Fähigkeit `cap_net_admin` gewähren (Rückfallweg der Kick-Funktion, nur nötig, wenn das Panel ohne den Proxy läuft)
3. fragen, welches Skript den Empfänger startet und welches ihn stoppt — darüber steuert das Panel den Server, und der Thermal Guard ebenso
4. fragen, wie weit der CPU-Überhitzungsschutz selbstständig gehen darf — `log`, `throttle`, `stop` oder `stop+restart` — und anbieten, `setup-cpufreq-perms.sh` auszuführen, damit die Throttle-Stufe ohne `root` funktioniert (siehe [THERMAL_GUARD.md](#))
5. `admin_config.json` mit allen Einstellungen schreiben
6. anbieten, zwei `systemd`-Units zu installieren (Panel + Proxy) — Start beim Booten und Neustart nach einem Absturz. **Empfohlen und voreingestellt** (bloßes Enter genügt): Der CPU-Überhitzungsschutz läuft nur, solange das Panel läuft — ohne die Units ist die Maschine nach einem Neustart ungeschützt. Der Schritt wird automatisch übersprungen, wenn `systemd` nicht PID 1 ist (Container, WSL1, OpenRC) oder kein `sudo` vorhanden ist.

Öffnen Sie nach der Einrichtung Ihren Browser über den Proxy:

```
http://YOUR_SERVER_IP:<proxy_port>/admin
```

Standardpasswort: `admin`

⚠ **Ändern Sie das Passwort sofort** — gehen Sie bei der ersten Anmeldung zu Settings. Ein Assistent beim ersten Start führt Sie durch das Festlegen von SDR-Verzeichnis, Prozessname, öffentlichem Port und neuem Passwort.

Schritt 3 — Starten / Stoppen / Neu starten

Es gibt zwei Wege, Panel und Proxy zu betreiben. **Wählen Sie einen und bleiben Sie dabei** — die Warnung am Ende dieses Abschnitts erklärt, was beim Mischen passiert.

Methode B (systemd) wird dringend empfohlen, wo immer `systemd` verfügbar ist. Der CPU-Überhitzungsschutz läuft im Panel: Ein Panel, das um 03:00 stirbt, oder eine Maschine, die neu startet, lässt die CPU ungeschützt, bis jemand sie von Hand wieder startet. `setup_admin.sh` bietet die Units jetzt standardmäßig an. Nutzen Sie Methode A, wenn es kein `systemd` gibt — Container, WSL1, OpenRC/sysvinit — oder wenn Sie die Überwachung bewusst selbst übernehmen wollen.

Methode A — `manage_admin.sh` (kein root, nichts zu installieren)

```
./manage_admin.sh start      # start admin panel and proxy
./manage_admin.sh stop       # stop both
./manage_admin.sh status     # show running status and PIDs
```

Es gibt bewusst **kein restart**. Früher beendete es das Panel und startete sofort seine eigene Kopie, was auf jeder Maschine mit Methode B mit `systemd` kollidiert. Für einen Neustart unter Methode A führen Sie `stop` und dann `start` aus — siehe [Panel neu starten](#) weiter unten.

Ein Skript steuert **beide** Prozesse. Es prüft vor dem Start des Proxys, ob `aiohttp` installiert ist, und gibt andernfalls eine klare Fehlermeldung mit dem Befehl zur Behebung aus.

Nichts startet von selbst: Nach einem Neustart oder nach einem Absturz des Panels starten Sie es wieder von Hand. Für einen Empfänger, den Sie beobachten, ist das in Ordnung — beachten Sie aber, dass der CPU-Überhitzungsschutz im Panel läuft: Solange das Panel unten ist, schützt nichts die Maschine (siehe [THERMAL_GUARD.md](#)).

Methode B — `systemd`-Units (Start beim Booten, Neustart nach Absturz)

Das Repository liefert zwei fertige Units mit, `phantomsdr-admin.service` und `phantomsdr-proxy.service`. Passen Sie in beiden `User=` und die Pfade an, dann:

```
sudo cp phantomsdr-admin.service phantomsdr-proxy.service /etc/systemd/system/
sudo systemctl daemon-reload
./manage_admin.sh stop      # free the ports first
sudo systemctl enable --now phantomsdr-admin
sudo systemctl enable --now phantomsdr-proxy
```

Die täglichen Befehle lauten dann:

```
sudo systemctl start    phantomsdr-admin
sudo systemctl stop     phantomsdr-admin
sudo systemctl restart  phantomsdr-admin
systemctl status        phantomsdr-admin
sudo journalctl -u phantomsdr-admin -f
```

Dieselben vier mit `phantomsdr-proxy`, oder beides zugleich: `sudo systemctl restart phantomsdr-admin phantomsdr-proxy`.

`admin.log` und `proxy.log` funktionieren genau wie bisher — die Units hängen an dieselben Dateien an.

Eines sollten Sie einmalig tun: die `tmpfiles`-Regel aus `tmpfiles/phantomsdr-logs.conf` installieren (zuvor Pfade und Benutzer anpassen). `systemd` legt diese beiden Logs als `root` an, bevor es auf `User=` wechselt, und der Knopf **Clear Logs** des Panels scheitert dann mit `Permission denied` — siehe unten „Clear Logs meldet Permission denied“. `setup_admin.sh` installiert die Regel für Sie.

Die Admin-Unit enthält `SupplementaryGroups=cpufreq`. Das erlaubt der Throttle-Stufe des Thermal Guard, den CPU-Takt zu senken, ohne dass das Panel als `root` läuft. Legen Sie diese Gruppe zuerst mit `./setup-cpufreq-perms.sh` an, sonst weigert sich `systemd`, die Unit zu starten; löschen Sie die Zeile, wenn Sie die Throttle-Stufe nicht nutzen.

Zurück zu Methode A:

```
sudo systemctl disable --now phantomsdr-admin
sudo systemctl disable --now phantomsdr-proxy
```

⚠ **Mischen Sie die beiden nicht.** Bei aktivierten Units wird `./manage_admin.sh stop` fünf Sekunden später von systemd rückgängig gemacht, und `./manage_admin.sh start` liefert Ihnen eine zweite, unverwaltete Kopie, die mit der von systemd um denselben Port streitet. (`restart` wurde genau deshalb aus dem Skript entfernt; es gibt jetzt einen Fehler aus und verweist auf `systemctl`.) `./manage_admin.sh status` liest nur und bleibt in beiden Fällen nützlich.

Panel neu starten

Im weiteren Verlauf dieses Dokuments steht an mehreren Stellen „Panel neu starten“. Gemeint ist je nach Methode:

```
# Methode B – systemd-Units (die normale Einrichtung)
sudo systemctl restart phantomsdr-admin phantomsdr-proxy

# Methode A – keine Units installiert
./manage_admin.sh stop && ./manage_admin.sh start
```

Die Unit bringt `KillMode=process` mit, und diese Zeile trägt Gewicht. Wenn Sie den SDR aus dem Panel starten, ist der Empfänger ein Kindprozess der Admin-Unit und erbt deren cgroup. Mit systemd's Standard `KillMode=control-group` würde ein Neustart des Panels spectrumserver, seinen Watchdog und den Autorun-Dienst mitnehmen — und vorher die vollen 90 Sekunden Stop-Timeout blockieren. Mit `KillMode=process` signalisiert systemd nur dem Panel selbst, sodass `sudo systemctl restart phantomsdr-admin` einen belegten Empfänger auf Sendung lässt.

Eine vor v4.1.0 installierte Unit nachrüsten. Ältere Unit-Dateien haben diese Zeile nicht, und ein Neustart des Panels fügt sie nicht hinzu: `systemctl restart` startet das Programm neu, ändert aber nicht dessen Konfiguration. Bearbeiten Sie die *installierte* Kopie — die Datei im Repository ist nur eine Vorlage, die systemd nie liest:

```
sudo nano /etc/systemd/system/phantomsdr-admin.service
```

Ergänzen Sie `KillMode=process` im Abschnitt `[Service]`, neben `Restart=always`. Danach systemd die Datei neu einlesen lassen und prüfen:

```
sudo systemctl daemon-reload
systemctl show phantomsdr-admin -p KillMode
```

Der zweite Befehl muss `KillMode=process` ausgeben. Steht dort weiterhin `control-group`, arbeitet systemd noch mit seiner zwischengespeicherten Kopie — `systemctl show phantomsdr-admin -p NeedDaemonReload` meldet `yes`, solange ein Reload aussteht. Führende Leerzeichen werden in Unit-Dateien ignoriert, die Einrückung spielt also keine Rolle.

Schritt 4 — Firewall-Port öffnen

Öffnen Sie in Ihrer Firewall nur den **öffentlichen Proxy-Port**. Der interne Port des Admin-Panels muss nicht nach außen freigegeben werden:

```
sudo ufw allow <proxy_port>
```

Schritt 5 — Kick-Funktion

Die Seite Users zeigt jeden aktuell verbundenen Hörer. Jede Zeile nennt die IP des Nutzers, die eingestellte Frequenz, die Betriebsart und die Verbindungsdauer. Um einen Nutzer zu trennen, klicken Sie in seiner Zeile auf **Kick** — nur diese eine TCP-Verbindung wird beendet. Alle anderen Hörer bleiben verbunden und bemerken nichts.

Der Kick ist sofort und punktgenau: Der Server startet nicht neu, für niemanden sonst wird der Ton unterbrochen, und der getrennte Nutzer kann sich sofort wieder verbinden (die Funktion dient dem Entfernen störender oder hängender Verbindungen, nicht dem Sperren).

Intern nutzt der Kick `ss -K dst <IP> dport <port>`, um den betreffenden TCP-Socket abzubauen. Das erfordert eine Linux-Fähigkeit, die das Einrichtungsskript automatisch gewährt. Falls Sie die Einrichtung übersprungen haben oder die Schaltfläche einen Fehler meldet, setzen Sie sie manuell:

```
sudo setcap cap_net_admin+ep $(which ss)
# Verify:
getcap $(which ss)    # should show: cap_net_admin=ep
```

Konfiguration — admin_config.json

Die gesamte Laufzeitkonfiguration liegt in `admin_config.json` in Ihrem PhantomSDR-Plus-Verzeichnis. `setup_admin.sh` schreibt die Erstfassung; spätere Änderungen sind über die Seite Settings oder durch direktes Bearbeiten der Datei möglich.

Wichtige von `setup_admin.sh` geschriebene Felder:

Schlüssel	Beschreibung
<code>port</code>	Interner Port des Admin-Panels (<code>admin_server.py</code> bindet hier)
<code>public_port</code>	Spectrumserver-Port (von der Seite Users genutzt, um <code>/users</code> abzufragen)
<code>proxy_port</code>	Öffentlicher Port, auf dem der Proxy lauscht
<code>sdr_host</code>	Host, über den der Proxy spectrumserver erreicht — <code>127.0.0.1</code> (siehe unten)
<code>password_hash</code>	SHA-256-Hash des Admin-Passworts
<code>sdr_base_dir</code>	Pfad zu Ihrer PhantomSDR-Plus-Installation
<code>sdr_process_name</code>	Zu überwachender Prozessname (Standard: <code>spectrumserver</code>)
<code>start_script</code> / <code>stop_script</code>	Skripte, mit denen das Panel den SDR-Server startet und stoppt — über sie handelt auch der thermische Schutz

Mehrere Empfänger auf einem Rechner? Setzen Sie `start_script` auf `start-all.sh`. Das Stoppskript hält bereits jeden Empfänger an, und Restart sowie der Temperaturschutz verwenden das Startskript; mit `start-all.sh` kehren deshalb alle Empfänger aus `receivers.toml` zurück statt nur des Hauptempfängers. Die Seite Users, die Hörerzahl und der Kick umfassen dann ebenfalls jeden Empfänger. Siehe [Mehrere Empfänger](#).

Schlüssel des thermischen Schutzes (alle optional — das Panel schreibt sie beim Speichern der Thermal-Guard-Einstellungen; fehlende Schlüssel verwenden diese Standardwerte):

Schlüssel	Standard	Beschreibung
thermal_enabled	true	Hauptschalter des Schutzes
thermal_mode	"log"	log (nur aufzeichnen, nie handeln) · throttle · stop · stop+restart
thermal_warn	null	Warntemperatur °C; null = aus dem kritischen Grenzwert Ihrer CPU ableiten
thermal_throttle	null	Temperatur, ab der der CPU-Takt gesenkt wird; null = automatisch
thermal_stop	null	Temperatur, ab der der Server gestoppt wird; null = automatisch
thermal_resume	null	Temperatur, die zur Erholung unterschritten werden muss; null = automatisch
thermal_sustain_s	60	Sekunden, die die Stopp-Temperatur gehalten werden muss
thermal_warn_sustain_s	30	Sekunden, die die Warn-/Throttle-Temperatur gehalten werden muss
thermal_resume_s	300	Sekunden unterhalb der Erholungstemperatur, bevor die Sperre aufgehoben wird
thermal_max_stops_hour	2	Thermische Stopps pro Stunde, bevor der automatische Neustart deaktiviert wird
thermal_test_temp	null	Fiktive Temperatur zum Testen; null = echten Sensor verwenden

Änderungen daran greifen bei der nächsten Messung — innerhalb weniger Sekunden, ohne Neustart.

Um Ports nach der Ersteinrichtung zu ändern, bearbeiten Sie `admin_config.json` und starten neu:

```
sudo systemctl restart phantomsdr-admin phantomsdr-proxy # B
./manage_admin.sh stop && ./manage_admin.sh start # A
```

Die Einstellung `sdr_host`

`proxy.py` verbindet sich mit `spectrumserver` über Loopback — `sdr_host` ist `127.0.0.1` in `admin_config.json`, dort von `setup_admin.sh` eingetragen. Sie sollten daran nichts ändern müssen.

Setzen Sie den Wert nur dann auf eine echte Adresse, wenn `spectrumserver` auf einer **anderen Maschine** läuft als der Proxy. Starten Sie danach das [Panel neu starten](#) neu.

Geändert: Frühere Versionen verlangten für `sdr_host` die LAN-IP der Maschine, weil `spectrumserver` WebSockets von der Loopback-Adresse schloss. Dieser Filter wurde entfernt, lokale Verbindungen funktionieren jetzt normal. Zwei Folgen: Der Aufruf von `http://localhost:<port>` auf der Server-Maschine selbst zeigt nun die vollständige Oberfläche (früher lud die Seite, blieb aber leer), und der Proxy fällt nicht mehr aus, wenn DHCP der Maschine eine neue IP zuweist. Wenn Sie aktualisieren und in Ihrer `admin_config.json` noch eine LAN-IP steht, ändern Sie sie auf `127.0.0.1` oder führen Sie `setup_admin.sh` erneut aus.

proxy.py — was es ist und wann Sie es brauchen

`proxy.py` legt sowohl den SDR-Server als auch das Admin-Panel auf einen **einzigsten öffentlichen Port** und leitet nach Pfadpräfix weiter:

Pfad	Weitergeleitet an
/admin*	Admin-Panel (localhost:<port>)
Alles andere	Spectrumserver (<sdr_host>:<public_port>)

Ohne den Proxy müssen Sie zwei Ports getrennt freigeben. Mit ihm geben Sie nur einen frei.

Der Proxy hebt außerdem die 4-MB-Grenze für WebSocket-Nachrichten auf (wichtig für große FFT-Rahmen des RX-888 bei 60 MSPS), leitet die echten Client-IPs über X-Forwarded-* -Header weiter und hält langlebige Verbindungen mit einem 30-Sekunden-Heartbeat durch NAT hindurch am Leben.

Ports nach der Einrichtung ändern

Bearbeiten Sie `admin_config.json` direkt:

```
{
  "port":      3000,
  "public_port": 9001,
  "proxy_port": 9002,
  "sdr_host":   "127.0.0.1"
}
```

Starten Sie dann beide Dienste neu:

```
sudo systemctl restart phantomsdr-admin phantomsdr-proxy # B
./manage_admin.sh stop && ./manage_admin.sh start      # A
```

Wenn `admin_server.py` beim Start zusätzlich an einen anderen Port binden soll (z. B. beim systemd-Dienst), können Sie das per Umgebungsvariable überschreiben:

```
ADMIN_PORT=3000 python3 admin_server.py
```

Standardmäßig bindet `admin_server.py` nur an `127.0.0.1`. Um es eigenständig ohne Proxy zu betreiben (Entwicklung/Test), binden Sie an alle Schnittstellen:

```
ADMIN_BIND=0.0.0.0 python3 admin_server.py
```

Nützliche manuelle Befehle

```
# Start admin panel manually (foreground)
python3 ~/PhantomSDR-Plus/admin_server.py

# Kill the admin panel
pkill -f admin_server.py

# Kill the proxy
pkill -f proxy.py

# Free a port that is stuck in use
sudo fuser -k 3000/tcp

# Check what is listening on a port
ss -tlnp | grep 3000
```

Protokolldateien

Datei	Inhalt
admin.log	Ausgabe des Admin-Panels
proxy.log	Startbanner des Proxys + eine Zugriffszeile je weitergeleiteter Anfrage
logwebsdr.txt	Log von Startskript und Watchdog — Start/Stopp, Empfängerstreiber, Registrierung bei websdr.org
spectrumserver.log	Die Ausgabe des Empfängerprozesses selbst — FFT- und OpenCL-Einrichtung, Verbindungen einzelner Clients, Fehler. Die größte dieser Dateien; sie wird rotiert, daher kann spectrumserver.log.1 daneben liegen
rade.log	Protokoll des RADE-/FreeDV-Sidecars
autorun.log	Protokoll des Decoder-Daemons — Spots, SNR, Drift
crash.log	Absturzberichte und thermische Ereignisse: [CRASH] -/ [TERMINATE] -Backtraces, [EXIT] -Notizen und [THERMAL] -Zeilen des Schutzes. Normalerweise leer — was darin steht, lohnt sich zu lesen

Alle sind über die Reiter der **Log Viewer** im Panel lesbar (LOGWEBSDR, SPECTRUMSERVER, ADMIN, RADE, CRASH, AUTORUN, PROXY). Der Viewer zeigt die letzten 150 Zeilen des gewählten Reiters und aktualisiert bei eingeschalteter Auto-Aktualisierung alle 3 Sekunden; er liest nur das Ende der Datei, deren Größe spielt also keine Rolle.

proxy.log und admin.log rotieren

proxy.log verzeichnet jede weitergeleitete Anfrage und admin.log jede Anfrage, die das Panel selbst erreicht. Keine von beiden wird standardmäßig rotiert. Früher dominierten die Abfragen des Dashboards beide Dateien — /admin/api/status läuft alle ~3 Sekunden, solange ein Tab offen ist —, doch diese Abfragen werden inzwischen aus beiden Protokollen herausgefiltert (siehe unten); die Rotation begrenzt daher langsames Wachstum statt einer Flut.

Das Repository liefert eine logrotate-Konfiguration unter logrotate/phantomsdr (täglich, oder früher, wenn ein Protokoll 10 MB überschreitet; behält 7 gzip-Archive in logproxy/, damit sie das Projektwurzelverzeichnis nicht zumüllen). **Sie ist standortspezifisch — öffnen Sie sie und passen Sie die drei Protokollpfade, den olddir-Pfad und den su-Benutzer an Ihre Maschine an, bevor Sie sie installieren**, dann:

```
sudo cp logrotate/phantomsdr /etc/logrotate.d/phantomsdr
sudo logrotate -d /etc/logrotate.d/phantomsdr # dry run, should list all three logs
```

logrotate/phantomsdr im Repository und /etc/logrotate.d/phantomsdr sind zwei **unabhängige Kopien** — cp verknüpft sie nicht. Änderungen an der Repository-Datei wirken sich auf einem laufenden System erst aus, wenn Sie das obige sudo cp erneut ausführen. Nur die Kopie in /etc liest logrotate tatsächlich.

Die Konfiguration verwendet copytruncate, was zwingend nötig ist: manage_admin.sh startet beide Prozesse mit >> <log>, und keiner öffnet seine Standardausgabe erneut; eine Rotation per Umbenennung ließe sie also weiter in die rotierte Datei schreiben, während das neue Protokoll für immer leer bliebe.

Archive landen in logproxy/ (von createolddir automatisch angelegt) und heißen proxy.log.1.gz ... proxy.log.7.gz, das neueste zuerst. Lesen Sie eines mit zcat logproxy/proxy.log.1.gz | less oder durchsuchen Sie alle auf einmal mit zgrep "pattern" logproxy/*.gz.

Die Schaltfläche **Clear Logs** des Panels leert **jede** Datei der obigen Tabelle, `autorun.log` und `proxy.log` eingeschlossen (`CLEAR_EXCLUDE` in `admin_server.py` ist leer). Beachten Sie, was das bei `autorun.log` kostet: Seine Dekodier-Historie ist der einzige Nachweis, was wann gehört wurde, und sie lässt sich nicht rekonstruieren — erhalten bleibt nur, was `logrotate` bereits nach `logproxy/` rotiert hat, ein Leeren verwirft also alles seit der letzten Rotation Dekodierte. Das Leeren bei laufendem Dekoder ist unbedenklich: Der Daemon hält die Datei mit `0_APPEND` offen und schreibt weiter in denselben Inode. Die Meldung nennt, was geleert wurde, und falls eine Datei nicht beschreibbar war — etwa ein `proxy.log`, das bei getrennten Benutzern einem anderen Benutzer gehört — nennt sie diese Datei und den Fehler, statt Erfolg zu melden, und lässt die Anzeige dieses Protokolls stehen, statt sie zu leeren. Die Dateien werden an Ort und Stelle geleert, nie gelöscht: Sowohl `manage_admin.sh` als auch die `systemd`-Units öffnen sie mit `0_APPEND`, sodass eine gelöschte Datei bis zum Neustart des Dienstes weiter einen unsichtbaren Inode füllen würde.

Erfolgreiche Abfragen werden aus **beiden** Zugriffsprotokollen herausgefiltert, vom `QuietPollFilter` in `proxy.py` und dem in `admin_server.py`. Das Dashboard ruft `/admin/api/status` alle 3 Sekunden auf sowie `/admin/api/logs`, `/admin/api/autorun/status`, `/admin/api/users`, `/admin/api/graph-stats` und `/admin/api/chat` alle 5 Sekunden; ungefiltert wären das ~95 % beider Dateien. Nur einfache 200er auf diesen Pfaden entfallen — jeder andere Status, Pfad oder jede andere Methode wird wie bisher protokolliert, sodass eine fehlschlagende Abfrage sichtbar bleibt und zustandsändernde Aktionen (`kick`, `autorun/start`, ...) über eigene Pfade laufen und stets aufgezeichnet werden. Die eine bewusste Ausnahme ist `/admin/api/logs/clear`, gefiltert in beiden Dateien: Seine eigene Zugriffszeile wird *nach* dem Leeren geschrieben, sodass ohne den Filter jedes erfolgreiche Leeren eine frische Zeile in genau der Datei hinterließ, die es gerade geleert hatte — und die Schaltfläche wirkte defekt.

Da `proxy.log` ein Zugriffsprotokoll ist, enthält es Besucher-IP-Adressen und User-Agent- Zeichenketten — denken Sie daran, bevor Sie es bei einer Hilfeanfrage weitergeben.

„Clear Logs“ meldet Permission denied

Auf einer Maschine, auf der die `systemd`-Units in einem frischen Verzeichnisbaum installiert wurden, kann der Knopf Folgendes zurückgeben:

```
x Could not clear admin.log ([Errno 13] Permission denied: /home/you/PhantomSDR-Plus/admin.log),
proxy.log ([Errno 13] Permission denied: /home/you/PhantomSDR-Plus/proxy.log)
```

Es sind immer diese beiden Dateien und nie die anderen fünf, denn nur sie werden von `systemd` selbst angelegt. `StandardOutput=append` wird von PID 1 geöffnet, **bevor** auf `User=` gewechselt wird; eine noch nicht vorhandene Logdatei entsteht daher als `root:root 0644`. Das Panel läuft als Ihr Benutzer und kann sie nicht leeren. Die anderen fünf Logs legen die Startskripte an, die ohnehin als Ihr Benutzer laufen, also lassen sie sich normal leeren. Wo die beiden Dateien älter sind als die Units — `manage_admin.sh` hat sie mit `>>` erzeugt —, gehören sie bereits dem richtigen Benutzer und der Knopf funktioniert; deshalb tritt das Problem nur bei einer Neuinstallation auf, die direkt mit `systemd` begonnen hat.

Prüfen Sie es mit `ls -l admin.log proxy.log`: die fehlschlagenden zeigen `root root`. Reparieren Sie sie:

```
sudo chown "$(id -un):$(id -gn)" admin.log proxy.log
sudo chmod 664 admin.log proxy.log
```

Ein Neustart ist nicht nötig — `systemd` schreibt weiter über den bereits geöffneten Dateideskriptor, und Anhängen ändert die Eigentümerschaft nicht. Drücken Sie **Clear Logs** erneut, und beide werden geleert.

Löschen Sie die Dateien nicht stattdessen. Beide sind mit `0_APPEND` geöffnet; wird eine entfernt, füllt `systemd` bis zum nächsten Dienstneustart einen unsichtbaren Inode, und die neu entstehende Datei gehört wieder `root`.

Damit es nicht wiederkommt, installieren Sie die tmpfiles-Regel aus dem Repository: `tmpfiles/phantomsdr-logs.conf`. Sie legt beide Logs bei jedem Bootvorgang mit dem richtigen Eigentümer an, bevor die Units starten, sodass eine entfernte Datei — manuelles Aufräumen, Neuinstallation, verschobenes Installationsverzeichnis — nie wieder root-eigen zurückkehrt. **Sie ist maschinenspezifisch: Öffnen Sie sie und passen Sie zuerst die beiden Pfade und `user:group` an**, dann:

```
sudo cp tmpfiles/phantomsdr-logs.conf /etc/tmpfiles.d/99-phantomsdr-logs.conf
sudo systemd-tmpfiles --create /etc/tmpfiles.d/99-phantomsdr-logs.conf
```

Wie bei `logrotate/phantomsdr` sind die Repo-Datei und die Kopie unter `/etc` unabhängig voneinander — das Bearbeiten der Repo-Kopie ändert nichts, bis Sie das `cp` erneut ausführen. Ab `systemd 254` repariert die Regel auch Eigentümer und Modus einer bereits vorhandenen Datei, behebt also den aktuellen Fehler mit; bei älteren Versionen ist das `chown` oben das, was den Knopf freischaltet.

`setup_admin.sh` erledigt all das beim Installieren der Units, eine mit dem Skript erstellte Einrichtung trifft dieses Problem also nie. Die Schritte oben gelten für ein von Hand installiertes Paar oder für eine Installation, die älter ist als der Log-Eigentümerschritt des Skripts.

Logrotate bringt das Problem nicht zurück: die mitgelieferte Konfiguration nutzt `copytruncate`, was denselben Inode und damit denselben Eigentümer beibehält.

Zugriffsübersicht

Aufbau	SDR	Admin-Panel
Ohne Proxy	<code>http://YOUR_IP:<public_port></code>	<code>http://YOUR_IP:<port>/admin</code>
Mit Proxy	<code>http://YOUR_IP:<proxy_port></code>	<code>http://YOUR_IP:<proxy_port>/admin</code>

Löschen von Chatnachrichten

Die Seite Chat History listet jede Nachricht des aktuellen Chatprotokolls auf. Jeder Eintrag hat eine Schaltfläche **Delete**, die genau diese eine Nachricht sofort aus dem Protokoll entfernt — ohne Serverneustart.

So funktioniert es:

- Das Admin-Panel schreibt die Chatprotokolldatei an Ort und Stelle neu und entfernt dabei nur die Zeile der gewählten Nachricht.
- Die Änderung wirkt für jeden Nutzer, der den Chat neu lädt; bereits geladener Chatverlauf in offenen Browser-Tabs wird nicht rückwirkend aktualisiert.
- Das Leeren des gesamten Protokolls (Schaltfläche **Clear All**) kürzt die Datei, was ebenfalls ohne Neustart wirkt.
- Eine **Chat-Antwort** wird als gewöhnliche Nachricht gespeichert, die mit einer Markierung beginnt, welche die beantwortete Nachricht benennt, zum Beispiel `[RE:2026-09-25 12:00:01|SV1BTL]`; in dieser Liste erscheint sie deshalb mit dieser Markierung davor. Das Löschen einer Antwort entfernt nur diese Antwort. Das Löschen der beantworteten Nachricht lässt ihre Antworten stehen; sie zeigen dann nur noch Name und Uhrzeit dessen, worauf sie antworten.

Die Löschschriftfläche wird in allen fünf Svelte-App-Varianten einheitlich dargestellt. Scheint eine Löschung nicht zu wirken, prüfen Sie, ob das Admin-Panel Schreibrechte auf die Chatprotokolldatei hat:

```
ls -l ~/PhantomSDR-Plus/chat.jsonl # path depends on your config
```

Wasserfall-Rundnachrichten

Das Panel **Waterfall Message** erlaubt es, ein dauerhaftes Textbanner in die Wasserfallanzeige jedes verbundenen Nutzers zu schieben, ohne den Serverprozess anzufassen.

Eine Nachricht senden

1. Öffnen Sie das Admin-Panel und gehen Sie zu **Waterfall Message**.
2. Geben Sie Ihren Nachrichtentext ein und wählen Sie eine Farbe (hexadezimal, z. B. `#ffdd00`).
3. Klicken Sie auf **Send** — die Nachricht erscheint sofort in allen aktiven Wasserfallansichten.

Die Nachricht löschen

Klicken Sie auf **Clear**, um das Banner aus allen Wasserfällen zu entfernen. Der Nachrichtenzustand wird vom Admin-Panel im Speicher gehalten; er wird automatisch gelöscht, wenn der Panel-Prozess neu startet.

Funktionsweise

Das Admin-Panel stellt zwei interne Endpunkte bereit, die der Proxy weiterleitet:

Endpunkt	Methode	Zweck
/admin/api/waterfall-message	POST	Aktuellen Bannertext und -farbe setzen oder löschen
/admin/api/waterfall-message	GET	Aktuellen Nachrichtenzustand als JSON zurückgeben

Das Frontend fragt den Nachrichtenzustand ab und rendert ihn als Overlay auf der Wasserfall-Zeichenfläche. Auf Clientseite ist weder eine WebSocket-Neuverbindung noch ein Neuladen der Seite nötig.

Typische Anwendungen

- Geplante Wartung ankündigen: "Server restart in 10 minutes"
- Bandbedingungen melden: "Solar flux 180 – 10m wide open"
- Begrüßung: "Welcome to SV1BTL WebSDR – Athens, KM17"

Systemgraphen

Die Seite **Graphen** zeichnet vier Messgrößen auf einer gemeinsamen Zeitachse:

- **CPU-Frequenz** — mittlerer Takt über alle Kerne, in GHz
- **CPU-Last** — Gesamtauslastung in Prozent
- **CPU-Temperatur** — in °C, mit gestrichelten Marken bei 70 °C (warm) und 80 °C (kritisch)
- **Nutzer online** — zu diesem Zeitpunkt mit dem WebSDR verbundene Hörer

Der Zeitraum wird mit **15 MIN / 1 HOUR / 4 HOURS / 12 HOURS / 24 HOURS** gewählt. Beim Überfahren mit der Maus erscheint ein Fadenkreuz mit allen vier Werten dieses Moments. Die vier Karten über dem Diagramm zeigen immer den neuesten Messwert.

Wie abgetastet wird

Ein Hintergrund-Thread in `admin_server.py` nimmt alle **2 Sekunden** einen Messwert. Die Werte liegen in zwei Stufen: die letzte **Stunde** in voller 2-Sekunden-Auflösung (1800 Punkte) und **24 Stunden** als 30-Sekunden-Mittelwerte (2880 Punkte) für die langen Zeiträume. Die Seite wählt die Stufe, die den gewählten Zeitraum abdeckt, und schreibt `30s averages` in die Kopfzeile, wenn die gemittelte Stufe angezeigt wird.

Ein ganzer Tag in 2-Sekunden-Auflösung wären ~43000 Punkte — zig MB Speicher, eine mehrere Megabyte große erste Übertragung und rund 40 Punkte je Bildpunkt, was kein Bildschirm darstellen kann. Die Mittelung auf 30 Sekunden hält einen Tag in 2880 Punkten.

Die Abtastung startet zusammen mit dem Admin-Panel, nicht erst beim Öffnen der Seite — die Seite zeigt also sofort bereits vorhandene Historie.

Der Puffer liegt **nur im Speicher**: nichts wird auf die Festplatte geschrieben, und die Historie geht beim Neustart des Admin-Panels verloren. Das ist Absicht — so bleibt die Funktion aus der Logrotation und von der Platte.

2 Sekunden ergeben eine nahezu Echtzeit-Ansicht und sind kurz genug, um die CPU-Frequenz nicht zu verfälschen: sie springt bei modernen CPUs von einem Messwert zum nächsten zwischen Leerlauf- und Turbotakt. Ein Messwert kostet etwa eine Millisekunde, der Sampler fällt neben dem SDR-Server also nicht ins Gewicht. Die Seite fragt im Takt der jeweiligen Stufe ab — alle 2 Sekunden bei den Live-Zeiträumen, alle 30 Sekunden bei den gemittelten, denn häufigeres Abfragen liefert nur leere Antworten.

Um Takt oder Vorhaltezeit zu ändern, passen Sie diese Konstanten am Anfang des Sampler-Blocks in `admin_server.py` an und starten Sie neu:

```
GRAPH_INTERVAL_S    = 2          # Sekunden zwischen den feinen Messwerten
GRAPH_FINE_S        = 3600       # Reichweite der feinen Stufe (1 h)
GRAPH_COARSE_EVERY  = 15         # feine Messwerte je grobem Punkt (15 x 2 s = 30 s)
GRAPH_COARSE_S      = 24 * 3600 # Reichweite der groben Stufe (24 h)
```

Woher die Werte stammen

Messgröße	Quelle
CPU-Frequenz	<code>psutil.cpu_freq()</code> , ersatzweise <code>/sys/devices/system/cpu/cpu*/cpufreq/</code> . Das sind dieselben Kerneldaten, die <code>cpufreq-info</code> aufbereitet — cpufrequtils wird nicht benötigt .
CPU-Last	<code>psutil.cpu_percent()</code>
CPU-Temperatur	<code>psutil.sensors_temperatures()</code> , ersatzweise <code>/sys/class/thermal</code> , dann der Befehl <code>sensors</code>
Nutzer online	Der <code>/users</code> -Endpunkt von <code>spectrumserver</code> selbst — die maßgebliche Sitzungsliste. Ein Zählen der Sockets mit <code>ss</code> sieht hinter <code>proxy.py</code> keine echten Client-IPs.

Ist eine Messgröße auf Ihrem Rechner nicht verfügbar, sagt das betreffende Feld das, statt eine Nulllinie zu zeichnen. Ein fehlgeschlagener Messwert unterbricht die Linie, damit eine Lücke nie wie ein gemessener Wert aussieht.

Warum vier Felder und nicht ein Diagramm

GHz, Prozent, Grad und eine Personenzahl haben keine gemeinsame Skala. In einem Diagramm bräuchte man mehrere y-Achsen, und die dabei entstehenden Schnittpunkte sind Artefakte der Skalierung, keine Aussagen über den Server. Vier gestapelte Felder auf einer Zeitachse halten den Vergleich ehrlich. Jedes Feld beginnt bei null, damit die Kurvenhöhe proportional zum Wert bleibt.

Farbe ist der Temperatur vorbehalten, wo Bernstein und Rot die oben genannten Schwellen markieren. Die übrigen Felder teilen sich eine Farbe, weil jedes nur eine Messreihe enthält, die sein Titel bereits benennt.

Endpunkt

GET /admin/api/graph-stats?since=<epoch> (Anmeldung erforderlich) liefert die Messwerte neuer als `since`, sodass die Seite inkrementell abfragt, statt bei jedem Takt den gesamten Puffer neu zu laden.

Thermal Guard

Vollständiges Handbuch: [THERMAL_GUARD.md](#) — die vier Modi und was ein Sysop bei jedem tun muss, Betrieb ohne Panel, die Throttle-Stufe ohne root, Testen und Fehlersuche.

Stoppt den SDR-Server, wenn die CPU gefährlich heiß wird, und startet ihn wieder, sobald sie abgekühlt ist. Er gehört zum Panel — nichts zusätzlich zu installieren — und erscheint als Karte **THERMAL GUARD** im Dashboard, mit seinen Einstellungen unter **Einstellungen**.

Ausgeliefert wird er **untätig**: Der Modus beginnt bei `log`, er zeichnet also zunächst nur auf, was er *getan hätte*. Nichts rührt Ihren Server an, bis Sie den Modus ändern.

Er funktioniert mit jeder Start-/Stopp-Methode

Der Schutz versucht nie herauszufinden, was Ihren Server überwacht. Er handelt über die Skripte, die Sie unter **Default start script** und **Default stop script** eingestellt haben, und solange die CPU überhitzt ist, wiederholt er den Stopp bei **jeder Prüfung** (alle 2 Sekunden). Alles, was den Server zurückbringt — der Watchdog in `start-*.sh`, eine systemd-Unit, ein Cron-Job, eine `tmux`-Sitzung — wird binnen weniger Sekunden rückgängig gemacht, bis die Maschine abgekühlt ist. Das ist die *Sperre* (Lockout).

Ist kein Stopp-Skript konfiguriert, sendet er `SIGTERM` an den in `sdr_process_name` genannten Prozess und nach 10 Sekunden Karenz `SIGKILL`.

Die Schwellen stammen aus Ihrer eigenen CPU

Statt einer festen Zahl, die auf den meisten Rechnern falsch wäre, liest der Schutz den kritischen Grenzwert, den der Kernel für Ihre CPU veröffentlicht, und rechnet von dort zurück:

Ihre CPU meldet	Warnung	Throttle	Stopp	Erholung
Intel, Grenze 100 °C	88	92	95	75
AMD Ryzen, Grenze 95 °C	83	87	90	70
Raspberry Pi, Grenze 85 °C	73	77	80	60

Das ist wichtig: Feste 95 °C sind auf einer Intel-CPU beinahe normal, auf einem Ryzen — dessen Tctl unter Boost konstruktionsbedingt bei 95 °C liegt — würden sie den Server auf einer gesunden Maschine stoppen, und auf einem Pi, der bei 80 °C drosselt, würden sie nie auslösen. Jede Schwelle lässt sich in den Einstellungen von Hand setzen; ein leeres Feld bedeutet *automatisch*.

So sehen Sie, wie Ihre Maschine dem Schutz erscheint:

```
python3 thermal_guard.py --once
```

```

sensor      : hwmon:coretemp
temperature: 58.0 C
crit        : 100.0 C
mode        : log
thresholds  : warn 88.0  throttle 92.0  stop 95.0  resume 75.0 (C)
cpufreq     : not writable – throttle stage disabled

```

Die vier Stufen

Den Server zu stoppen ist das letzte Mittel, nicht das einzige:

1. **warn** — wird protokolliert, sonst nichts.
2. **throttle** — senkt die CPU-Taktobergrenze um 20 %. Erfordert einen cpufreq-Treiber *und* Schreibrechte auf `scaling_max_freq`, also in der Regel root; wo das fehlt (der Normalfall für ein Panel ohne Privilegien und auf den meisten VPS), **deaktiviert sich die Stufe selbst** und der Schutz geht von warn direkt zu stop. `./setup-cpufreq-perms.sh` erteilt diese Schreibrechte einer Gruppe `cpufreq`, sodass die Stufe ohne root funktioniert — siehe [THERMAL_GUARD.md](#).
3. **stop** — führt Ihr Stopp-Skript aus und hält den Server unten (siehe Sperre).
4. **resume** — nachdem die CPU `thermal_resume_s` lang (standardmäßig 5 Minuten) unter der Erholungstemperatur geblieben ist, wird die Taktgrenze wiederhergestellt und im Modus `stop+restart` Ihr Start-Skript erneut ausgeführt.

Jede Stufe muss ihre Haltezeit **durchhalten** — 30 s für warn und throttle, 60 s für stop —, damit eine kurze Spitze beim Kompilieren nie auslöst.

Wiederholte Auslösungen begrenzt `thermal_max_stops_hour` (Standard 2). Danach wird der *automatische Neustart* deaktiviert, während der Schutz weiterarbeitet, damit eine überhitzende Maschine nicht zwischen Laufen und Gestopptsein pendelt.

Welche Sensoren verwendet werden

Nur echte CPU-Die-Sensoren: `coretemp` (Intel), `k10temp` / `zenpower` (AMD), `cpu_thermal` / `soc_thermal` (ARM, Raspberry Pi). Verwendet wird der höchste Wert des Chips, nicht der Durchschnitt.

`acpitz` und unbeschriftete Thermal Zones werden **bewusst ignoriert** — sie melden oft eine Gehäuse- oder Platinentemperatur, die zig Grad unter der tatsächlichen CPU liegt; ein Schutz, der ihnen vertraut, würde nie auslösen, wenn es darauf ankommt. Hat Ihre Maschine keinen brauchbaren Sensor (häufig auf einem VPS oder in einem Container), sagt die Karte das offen (`no trusted CPU sensor`), statt Schutz vorzutäuschen.

Was er schreibt

Alles landet in `crash.log`, also im Reiter **CRASH** des Log Viewers, ein Ereignis pro Zeile:

```

[THERMAL] warn: 89.0C (warn=88.0 crit=100.0) sustained 31s
[THERMAL] throttle: 93.0C sustained 30s – cpufreq max lowered 20%
[THERMAL] STOPPED server: 96.0C sustained 62s (stop=95.0 crit=100.0) – Started stop-websdr.sh
[THERMAL] lockout: process reappeared at 94.0C – re-stopped [3 time(s) so far]
[THERMAL] recovered: 74.0C held below 75.0 – lockout cleared
[THERMAL] auto-restart: Started start-rx88mk2.sh

```

Die Sperr-Zeile wird auf eine pro Minute begrenzt, damit ein Supervisor, der sich dauernd wehrt, das Log nicht überschwemmt.

Testen, bevor Sie ihm vertrauen

Setzen Sie **TEST TEMPERATURE** in den Einstellungen auf einen Wert oberhalb Ihrer Stopp-Schwelle. Der Schutz behandelt ihn wie eine echte Messung, sodass der ganze Weg — warn, throttle, stop, Sperre, Erholung, Neustart — auf Wunsch abläuft. Feld leeren, um zum echten Sensor zurückzukehren.

Führen Sie diesen Test im Modus `stop` durch, wird der Server tatsächlich gestoppt und Ihre Hörer werden getrennt. Machen Sie das, wenn niemand zuhört, oder lassen Sie den Modus auf `log`, wo der Test zeigt, was passiert *wäre*, ohne etwas anzufassen.

Empfohlene Einführung

1. Lassen Sie den Modus eine Woche auf `log` und arbeiten Sie normal weiter.
2. Lesen Sie danach den CRASH-Reiter nach den harten Momenten — einem vollen Rebuild, einem heißen Nachmittag. Steht dort nichts, kam Ihre Maschine nie in die Nähe.
3. Sehen die protokollierten Temperaturen stimmig aus, stellen Sie den Modus auf `stop` (oder `stop+restart`, damit er von selbst zurückkommt).
4. Bestätigen Sie es einmal mit dem Testtemperatur-Feld und lassen Sie es dann in Ruhe.

Eine Sperre aufheben

CLEAR LOCKOUT auf der Dashboard-Karte hebt Sperre und Ratenbegrenzung auf. Es entschärft den Schutz nicht: Ist die CPU weiterhin zu heiß, stoppt die nächste Prüfung den Server einfach wieder. Verwenden Sie es, nachdem Sie die Kühlung in Ordnung gebracht haben.

Endpoint

GET `/admin/api/thermal` (Anmeldung erforderlich) liefert den vollständigen Zustand — Sensor, Schwellen, Stufe, Sperre, letztes Ereignis. Ein POST mit `{"action": "reset"}` entspricht der Schaltfläche CLEAR LOCKOUT.

Ohne das Admin-Panel

`thermal_guard.py` kommt mit der Standardbibliothek aus und läuft auch allein, mit derselben `admin_config.json`:

```
python3 thermal_guard.py --once           # Sensor, Grenzwert und Schwellen anzeigen
python3 thermal_guard.py                 # als Dienst betreiben (nur Protokoll, bis
scharfgeschaltet)
python3 thermal_guard.py --mode stop+restart # scharfschalten, ganz ohne Konfigurationsdatei
python3 thermal_guard.py --config /path/to/other.json
python3 thermal_guard.py --help
```

`--mode` überschreibt `thermal_mode` aus der Konfigurationsdatei — eine Maschine ohne Panel lässt sich also direkt aus ihrer systemd-Unit scharfschalten, ohne jedes JSON.

Spot Reporting (Autorun FT8/FT4/JS8/WSPR)

Der Reiter **Spot Reporting** steuert den Autorun-Daemon (`autorun/index.js`) — einen Node.js-Prozess, der FT8/FT4/JS8/WSPR serverseitig direkt vom Empfänger dekodiert und die Spots zu den Reporting-Netzwerken hochlädt:

- **FT8 / FT4 / JS8 → PSK Reporter** (IPFIX/UDP)
- **WSPR → wsprnet.org** (HTTP)

JS8 wird nur mit Normal-Geschwindigkeit gespottet — dem 15-s-Anrufzyklus, auf dem Heartbeats und CQs laufen. Die Spots stammen aus Heartbeats, Compound-Frames und gerichteten Nachrichten; Gruppenziele wie @ALLCALL und vom Decoder nicht aufgelöste Rufzeichen (<...>) werden nie gemeldet. JS8 teilt sich die PSK-Reporter-Warteschlange mit FT8 und FT4 und wird genauso getrennt gezählt.

Das Reporting ist standardmäßig AUS. Es wird nichts hochgeladen, bis Sie ein Ziel aktivieren und **Start** drücken. Dekodieren und Hochladen sind unabhängig voneinander: Der Daemon kann bei ausgeschaltetem Reporting dekodieren und protokollieren, sodass Sie die Aktivität prüfen können, bevor etwas öffentlich wird.

△Spots werden unter **Ihrem Rufzeichen** in öffentliche Netze hochgeladen. Aktivieren Sie nur Bänder/Betriebsarten, die Ihr Empfänger wirklich hört, und tragen Sie den korrekten Locator ein.

Voraussetzungen

Der Autorun-Daemon benötigt Node.js 22+, die npm-Pakete `ws` + `cbor-x` (aufgelöst über den Symlink `autorun/node_modules` → `frontend/node_modules`) sowie `util-linux` (`taskset`). Die Installer richten all das automatisch ein — siehe den [Abschnitt Autorun-Spot-Reporter in INSTALLATION.md](#). Schlägt **Start** fehl, ist meist dieser Symlink oder `taskset` die Ursache (siehe Fehlerbehebung unten).

Den Reiter verwenden

1. **Identität** — Rufzeichen und Locator. Vorausgefüllt aus `frontend/site_information.json` (`siteSysop` / `siteGridSquare`, auf 6 Zeichen gekürzt); hier bei Bedarf überschreiben.
2. **Band-x-Betriebsart-Matrix** — kreuzen Sie die zu dekodierenden Kombinationen an. Zeilen sind Bänder, Spalten FT8 / FT4 / JS8 / WSPR; nicht unterstützte Felder sind deaktiviert. WSPR deckt zusätzlich die LF-/MF-Bänder (2200 m, 630 m) und einen zusätzlichen 80-m-EU-Kanal ab.
3. **Ziele** — aktivieren Sie **PSK Reporter** und/oder **wsprnet** (beide standardmäßig aus).
4. **Max slots** — eine Sicherheitsgrenze dafür, wie viele Band-/Betriebsart-Slots gleichzeitig laufen dürfen.
5. **Start / Stop** — startet/beendet den Daemon (per `taskset` an die obersten CPU-Kerne gebunden, automatisch für die Maschine gewählt — siehe *Ressourcenbedarf und Grenzen* unten). **Save / Reload** speichern und lesen die Konfiguration neu; **Free All** leert alle Slots.

Die Statuskarte aktualisiert sich alle 5 s und zeigt Laufzustand, Dekodier-/Uploadzähler und den letzten Uploadzeitpunkt. Ein Abzeichen **REPORTING** erscheint im Hauptwasserfall, solange das Reporting aktiv ist.

„0 sent“ in den ersten Minuten ist normal. PSK Reporter lädt gebündelt alle **5 Minuten** hoch, **wsprnet** alle **2 Minuten** — Spots warten bis zum nächsten Absenden. Die Statuskarte zeigt die Anzahl der wartenden Spots und die Taktung.

Bänder oder Modi im laufenden Betrieb ändern

Änderungen wirken erst nach einem Neustart des Daemons. `autorun/index.js` liest `autorun.json` genau einmal, beim Start. Es gibt keine Dateiüberwachung und kein Reload-Signal — behandelt werden nur SIGINT und SIGTERM, die beide „beenden“ bedeuten. Ein laufender Daemon decodiert deshalb weiter die Slots, mit denen er gestartet wurde, ganz gleich was Sie danach speichern.

So übernehmen Sie eine Änderung:

1. Bänder und Modi an- bzw. abwählen
2. **SAVE CONFIG**

3. STOP

4. START

Oder einfach **STOP** → Auswahl ändern → **START**, denn START speichert die Konfiguration vor dem Start.

⚠ **START ohne vorheriges STOP übernimmt die Änderung nicht.** START speichert die Konfiguration, dann antwortet die Start-Anfrage mit `already running` und es wird nichts neu gestartet. Sie sehen eine „gespeichert“-Meldung neben einem „already running“-Fehler, während der Daemon mit den **alten** Bändern weiterläuft — das sieht leicht nach Erfolg aus. Immer zuerst STOP.

Was ein Neustart mit den Zählern macht:

- **Die Gesamtzahlen neben den Kontrollkästchen bleiben erhalten** — sie stehen in `autorun-totals.json`, die beim Beenden nicht gelöscht wird.
- **Die Decoder-Kacheln werden auf null zurückgesetzt**, sie gelten je Lauf.
- **Nichts aus der Warteschlange geht verloren**: beim Beenden werden offene Spots noch gesendet.
- Ein **neu aktiviertes** Band beginnt bei 0; ein **zuvor genutztes** Band setzt seine Summe fort.
- Ein **abgewähltes** Band behält seine Zahl neben dem nun leeren Kästchen; die Historie wird nicht gelöscht.

Die Decodierung pausiert nur für die wenigen Sekunden zwischen STOP und START.

Spot-Zähler

Es werden zwei verschiedene Zählungen angezeigt, die unterschiedliche Fragen beantworten.

SPOTS UPLOADED PER DECODER — eine Kachel je Decoder (FT8 / FT4 / JS8 / WSPR) mit den Spots, die **seit dem letzten Start des Daemons** hochgeladen wurden, darunter die Zahl der Decodierungen und wie viele noch in der Warteschlange stehen. Stop/Start setzt diese Werte auf null zurück. Ein Decoder, dessen Ziel abgeschaltet ist, zeigt seine Decodierungen und `reporting off` statt einer nackten 0 — dort sind null Uploads eine Einstellung, kein Fehler.

Die Zahl neben jedem Kontrollkästchen unter BANDS & MODES ist die Zahl der **insgesamt** hochgeladenen Spots dieser Band-/Modus-Kombination. Sie steht in `autorun-totals.json` und übersteht Neustarts. Ein vorangestellter Punkt (· 123) in Bernstein bedeutet, dass dieser Slot decodiert, aber noch nichts hochgeladen hat. Zwischen zwei Uploads ist das normal — PSK Reporter sendet alle 5 Minuten, wspnet alle 2 —, daher stehen FT8/FT4-Zähler nach einem Start die ersten Minuten in Bernstein. Bernstein bleibt es auch, wenn das Ziel dieses Decoders abgeschaltet ist. Eine schlichte 0 ohne Punkt bedeutet, dass der Slot aktiviert ist, aber noch nichts decodiert hat — WSPR steht nach einem Start mehrere Minuten auf null, weil es im 2-Minuten-Takt läuft, während FT8 schon in die Hunderte zählt. Ein Slot, den der Daemon nie ausgeführt hat, zeigt gar nichts. Der Tooltip zeigt Uploads, Decodierungen und den Zeitpunkt des letzten Uploads.

FT8 und FT4 teilen sich eine PSK-Reporter-Warteschlange, deshalb zählt der Daemon jeden Spot beim Versand seinem eigenen Modus und Band zu; die Aufteilung wird nicht nachträglich aus den Summen geschätzt.

Zähler zurücksetzen. FREE ALL SLOTS wählt alle Bänder/Modi ab, schaltet beide Ziele aus, stoppt den Daemon und **löscht die Gesamtzähler** — danach steht neben keinem Kontrollkästchen mehr eine Zahl. Das ist die einzige Möglichkeit, sie zurückzusetzen, und sie lässt sich nicht rückgängig machen. Die Schaltfläche stoppt den Daemon und wartet auf dessen Ende, bevor sie `autorun-totals.json` löscht, denn der Daemon schreibt diese Datei beim Beenden neu; ein Löschen bei laufendem Daemon würde die Zahlen einfach zurückholen.

Ressourcenbedarf und Grenzen

Der Daemon ist bewusst schlank und läuft auf ressourcenarmer Hardware (ein 4-Kern-i5 genügt), es gibt aber reale Grenzen, die man kennen sollte.

Das CPU-Pinning von Autorun ist konfigurationsbewusst und automatisch. Der **Autorun-Daemon** wird stets vom Admin-Panel (`admin_server.py`) gestartet, sein Pinning wird daher bei jeder Installation ohne Konfiguration gesetzt: Aus der Anzahl der CPUs wird ein `taskset`-Kernbereich abgeleitet, der einige der obersten Kerne für das Dekodieren reserviert; bei ≤ 4 Kernen läuft er ungebunden.

Das Pinning von spectrumserver liegt bei Ihrer eigenen Startmethode. Wie Sie spectrumserver starten, unterscheidet sich je nach Installation (SDR-Typ, Werkzeuge, eigene Skripte), daher setzt dieses Dokument keinen bestimmten Starter voraus. Das Admin-Panel startet und bindet spectrumserver nicht — das hängt ganz vom Befehl oder Dienst ab, mit dem Sie ihn ausführen. Wollen Sie spectrumserver von den Kernen fernhalten, die der Autorun-Daemon nutzt, binden Sie ihn selbst mit `taskset` in Ihrem Startbefehl (siehe Abschnitt zum Überschreiben unten). Tun Sie das nicht, läuft er einfach ungebunden und der Scheduler des Betriebssystems verteilt ihn — korrekt und sicher, nur ohne die bewusste Kerntrennung.

Zur Orientierung reserviert der Autorun-Daemon diese oberen Kerne (wenn Sie spectrumserver binden, bleiben Sie also bei den unteren Kernen, um Überschneidungen zu vermeiden):

Logische CPUs	Vom Autorun-Daemon genutzt	Für spectrumserver frei lassen
≤ 4	<i>ungebunden</i>	<i>ungebunden</i>
6	5	0-4
8	6-7	0-5
12 (z. B. Hybrid 8P+4E)	8-11	0-7
16	12-15	0-11

Auf einer Maschine mit **4 Kernen (oder weniger)** gibt es nichts zu trennen, daher läuft auch der Autorun-Daemon *ungebunden* und teilt sich alle Kerne — korrekt und sicher, aber die SDR-FFT und die Dekodierschübe konkurrieren um dieselben Kerne.

Bekannte Grenzen:

- Der eigentliche Engpass ist spectrumserver + die SDR-Bandbreite, nicht der Daemon.** Ein RX888 bei 30 MHz braucht OpenCL/GPU; eine CPU mit wenigen Kernen ohne leistungsfähige GPU kann diese FFT nicht tragen. Kombinieren Sie schwache Hardware mit einem schmaleren SDR (RSP1A ≈ 10 MHz, RTL-SDR ≈ 2.4 MHz).
- WSPR ist der CPU-Flaschenhals.** Sein Fano-Decoder ist eine JS-Portierung (~ 30 s pro Band, einfädig). Der 4-Worker-Pool führt 4 Dekodierungen parallel aus; mehr als **etwa 4 WSPR-Bänder** gleichzeitig können die Warteschlange über den 120-s-Zeitschlitz hinaus verlängern, besonders während das SDR um Kerne konkurriert. FT8/FT4-Dekodierungen sind im Vergleich günstig.
- Skalierung der Slot-Anzahl.** Die Angabe von ~ 2 – $2,5$ Kernen / 600–700 MB gilt für die vollen ~ 28 Slots auf einer 8-Kern-Maschine. Auf 4 gemeinsam genutzten Kernen bleiben Sie bei wenigen Bändern (Richtwert: ≤ 6 FT8/FT4 + ≤ 3 WSPR) und beobachten die `queued`-Statistik des Pools.
- Das Pinning setzt eine hybride Intel-Topologie voraus** (untere Kerne = schnellere P-Cores). Auf AMD oder CPUs mit verschränkter SMT-Nummerierung bleibt die automatische Aufteilung gültig (keine Überschneidung, kein Absturz), aber „untere Kerne sind schneller“ trifft womöglich nicht wörtlich zu; bei einem 4C/8T-Modell mit Hyperthreading sind die oberen Kerne SMT-Geschwister — eingegrenzt, nicht vollständig isoliert. Wo der automatische Bereich nicht ideal ist, nutzen Sie das **manuelle Überschreiben** unten.

5. **Die Bandabdeckung wird vom Empfänger begrenzt.** Die RX888-Konfiguration (`sps=60000000` → Nyquist bei 30 MHz) erreicht das 6-m-Band und darüber nicht; die Bandtabelle reicht von 160 m bis 10 m. Andere SDRs decken ab, was ihr Abstimmfenster erlaubt.
6. **Reporting-Taktung.** PSK Reporter sendet alle 5 min, wsprnet alle 2 min — „0 sent“ in den ersten Minuten ist normal, kein Fehler.

Manuelles Überschreiben des CPU-Pinnings (fortgeschritten)

Die automatisch abgeleitete Kernaufteilung passt für die meisten Maschinen, aber Sie können dort ein bestimmtes Pinning erzwingen, wo sie es nicht tut (AMD CCX/CCD, ARM big.LITTLE, verschränktes SMT). Zwei unabhängige Überschreibungen, jede akzeptiert eine Kernliste im `taskset -c`-Format (2-3, 0,2,4, 0-2,5) oder `none/off/unpinned`, um das Pinning abzuschalten. Ein ungültiger Wert wird ignoriert und die automatische Ableitung greift — ein Tippfehler kann einen Start also nie verhindern.

Autorun-Daemon — Umgebungsvariable `AUTORUN_CORES` oder ein Feld `"cores"` in `autorun.json` (die Umgebung gewinnt). Das Feld `"cores"` bleibt beim **Save** im Admin-Panel erhalten, eine Handänderung bleibt also bestehen. Zwei Wege, es zu setzen:

Variante A — `autorun.json` (dauerhaft, empfohlen). Fügen Sie der Konfiguration im Repository-Wurzelverzeichnis eine Zeile `"cores"` hinzu und führen Sie dann im Reiter Spot Reporting **Stop → Start** aus:

```
// autorun.json
{ "identity": { "callsign": "SV1BTL", "grid": "KM17VX" },
  "reporting": { "pskreporter": true, "wsprnet": false },
  "slots": [ /* ... */ ],
  "cores": "2-3" }           // or "none" to run unpinned
```

Variante B — Umgebungsvariable (einmalig). Der Daemon wird vom Admin-Panel gestartet, die Variable muss also in der Umgebung **des Panels** stehen — setzen Sie sie und starten Sie das Panel neu (die Umgebung gewinnt gegenüber dem Wert in `autorun.json`):

```
# Methode A – die Variable muss auf dem startenden Prozess gesetzt sein:
./manage_admin.sh stop && AUTORUN_CORES=2-3 ./manage_admin.sh start

# Methode B – systemd ignoriert eine auf der systemctl-Kommandozeile
# gesetzte Variable, legen Sie sie daher in der Unit ab:
sudo systemctl set-environment AUTORUN_CORES=2-3
sudo systemctl restart phantomsdr-admin
```

spectrumserver — binden Sie ihn selbst, indem Sie `taskset -c <Kerne>` dem Befehl voranstellen, mit dem Sie den Server starten. Das funktioniert mit jeder Startmethode (direkter Aufruf, SDR-Pipeline, Service-Unit usw.):

```
# pin the server to cores 0-3, direct launch:
taskset -c 0-3 ./build/spectrumserver --config <your-config>.toml &lt; <your-input>
# or in an SDR pipeline:
<your-sdr-source> | taskset -c 0-3 ./build/spectrumserver --config <your-config>.toml
```

⚠ **Sorgen Sie dafür, dass es Neustarts übersteht.** Ein `taskset` in der Befehlszeile gilt nur für diesen einen Start. Startet etwas den Server automatisch neu (ein Watchdog, ein systemd-Dienst, ein `cron-/@reboot`-Eintrag ...), fügen Sie das `taskset`-Präfix in das Skript oder die Unit ein, die ihn tatsächlich startet — sonst geht das Pinning beim Neustart verloren.

Prüfen, ob das Pinning gegriffen hat — die tatsächliche CPU-Affinität der laufenden Prozesse ausgeben:

```
taskset -cp "$(pgrep -f 'autorun/index.js')" # autorun daemon
taskset -cp "$(pgrep -x spectrumserver)" # spectrumserver
```

Zulässige Werte (beide Überschreibungen): eine `taskset -c`-Liste (2-3, 0,2,4, 0-2,5) oder `none/off/unpinned` für kein Pinning. Alles Fehlerhafte wird ignoriert und die automatische Ableitung greift, ein Tippfehler kann den Start also nie blockieren.

Auf einem Raspberry Pi / jeder Maschine mit ≤ 4 Kernen brauchen Sie in der Regel keines von beidem — der automatische Weg lässt beide Prozesse bereits ungebunden laufen, was auf einer kleinen, homogenen CPU die richtige Wahl ist. Das Überschreiben ist für größere Maschinen gedacht, die keine Intel-Hybriden sind.

Dateien und Endpunkte

Element	Zweck
<code>autorun.json</code>	Gespeicherte Konfiguration (Identität, Slots, Ziele, max. Slots, optionales cores-Pinning). Vom Reiter geschrieben; von git ignoriert.
<code>autorun-status.json</code>	Live-Status, den die Statuskarte liest (PID, Zähler, letzter Upload). Alle 15 s geschrieben; beim Stoppen entfernt.
<code>frontend/dist/autorun-active.json</code>	Öffentliche Abzeichendaten, ausgeliefert unter <code>/autorun-active.json</code> . Leer, wenn das Reporting aus ist.
<code>autorun.log</code>	Standard- und Fehlerausgabe des Daemons.
GET/POST <code>/admin/api/autorun/config</code>	<code>autorun.json</code> lesen/schreiben (prüft Rufzeichen, Locator, Band-/Betriebsart-Kombinationen, Slot-Grenze).
GET <code>/admin/api/autorun/status</code>	Laufzustand (pgrep) + <code>autorun-status.json</code> .
POST <code>/admin/api/autorun/start / stop</code>	Startet (<code>taskset -c <auto> node autorun/index.js</code>) / sendet SIGTERM. Der Kernbereich wird von der CPU abgeleitet (siehe oben).

Hinweis: Nach einem Upgrade von `admin_server.py` müssen Sie das [Panel neu starten](#) neu starten, um neue Autorun-Routen oder eine aktualisierte Band-/Betriebsart-Matrix zu laden. Der Daemon läuft als eigener Prozess und übersteht daher Neustarts des Admin-Panels.

Fehlerbehebung

Problem	Abhilfe
Admin-Panel startet nicht	Prüfen Sie <code>admin.log</code> — meist ein fehlendes Python-Paket
Proxy startet nicht	Führen Sie <code>python3 -c "import aiohttp"</code> aus — schlägt es fehl: <code>pip3 install aiohttp --break-system-packages</code>
<code>proxy.log</code> ist leer	Prüfen Sie zuerst, ob der Proxy wirklich läuft (<code>./manage_admin.sh status</code>). Wenn ja, nutzen Sie eine alte Version: Der Starter muss <code>python3 -u</code> verwenden (ungepuffert — sonst verlässt das Startbanner nie den 8-KB-Puffer), und <code>main()</code> in <code>proxy.py</code> muss <code>logging.basicConfig()</code> aufrufen (sonst hat aiohttps Zugriffslogger keinen Handler und jede Anfragezeile geht verloren, weil <code>web.AppRunner</code> — anders als <code>web.run_app</code> — die Protokollierung nicht konfiguriert).
Proxy startet, aber der Wasserfall bleibt leer	Prüfen Sie <code>sdr_host</code> in <code>admin_config.json</code> — normalerweise <code>127.0.0.1</code> . Steht dort noch eine alte LAN-IP aus einer früheren Version, ändern Sie sie und starten Sie das Panel neu starten neu
Kick-Schaltfläche meldet „no connections“	Führen Sie <code>getcap \$(which ss)</code> aus — fehlt <code>cap_net_admin</code> , setzen Sie <code>sudo setcap cap_net_admin+ep \$(which ss)</code>
Status zeigt immer OFFLINE	Gehen Sie zu Settings → SDR Process Name — tragen Sie den exakten Namen ein, den <code>ps -eo comm,args \ grep -v grep</code> anzeigt
Seite Users zeigt keine Daten	Prüfen Sie den Endpunkt: <code>curl http://127.0.0.1:<public_port>/users</code>
Externer Browser meldet NetworkError	Prüfen Sie, ob der Proxy läuft: <code>./manage_admin.sh status</code>
Proxy nach einer Netzwerkänderung defekt	Nur relevant, wenn <code>sdr_host</code> noch eine LAN-IP enthält — setzen Sie ihn auf <code>127.0.0.1</code> und starten Sie das Panel neu starten neu
Chat-Löschschaltfläche wirkt nicht	Prüfen Sie die Schreibrechte auf die Chatprotokolldatei: <code>ls -l ~/PhantomSDR-Plus/chat.jsonl</code>
Wasserfallnachricht erscheint nicht	Prüfen Sie, ob der Proxy läuft (<code>./manage_admin.sh status</code>) und das Frontend auf einem aktuellen Build mit Overlay-Renderer ist
Wasserfallnachricht nach Neustart verschwunden	Erwartet — der Nachrichtenzustand liegt nur im Speicher; senden Sie sie nach dem Neustart des Admin-Panels erneut
„Start“ bei Spot Reporting schlägt fehl / Daemon beendet sich sofort	Prüfen Sie <code>autorun.log</code> . Meist fehlt der Symlink <code>autorun/node_modules</code> (<code>ln -sf ../frontend/node_modules autorun/node_modules</code>) oder <code>taskset</code> ist nicht installiert (<code>sudo apt install -y util-linux</code>).
Spot Reporting: Daemon läuft, aber <code>decodes</code> bleibt 0 und <code>autorun.log</code> zeigt 504 / tap closed 1006	Der Audioabgriff erreicht <code>spectrumserver</code> nicht. Der Daemon erkennt den Port automatisch aus der Konfiguration des laufenden Servers; die Protokollzeile <code>[autorun] tap backend: HOST:PORT</code> muss zu Ihrem <code>[server] port</code> passen. Stimmt sie nicht (oder lief der Server beim Start nicht), fixieren Sie ihn in <code>autorun.json</code> : <code>"server": { "host": "127.0.0.1", "port": 9002 }</code> , dann Stop→Start.
Spot Reporting zeigt „0 sent“	In den ersten Minuten normal — PSK Reporter sendet alle 5 min, wsprnet alle 2 min. Prüfen Sie, ob ein Ziel aktiviert ist und der Daemon läuft.
Neue Bänder/Betriebsarten erscheinen nicht in der Matrix	Panel neu starten — die Matrix wird beim Start geladen.
<code>autorun.log</code> zeigt <code>MODULE_TYPELESS_PACKAGE_JSON</code> /	Kosmetische Warnung (die Dekodierung funktioniert weiterhin). In <code>frontend/src/modules/package.json</code> fehlt <code>"type": "module"</code> ;

Problem	Abhilfe
„Reparsing as ES module ... performance overhead“	fügen Sie die Zeile nahe dem Anfang ein und führen Sie Stop→Start aus. Kein Frontend-Neubau nötig — der Daemon importiert die Quelldatei direkt.
„Start“ bei Spot Reporting scheitert mit <code>taskset: ... Invalid argument</code> auf einem alten/kleinen PC	Sollte mit dem konfigurationsbewussten Starter nicht vorkommen — der Kernbereich wird aus der CPU-Anzahl abgeleitet und läuft bei ≤ 4 Kernen ungebunden. Tritt es auf, ist Ihr <code>admin_server.py</code> älter als diese Änderung; aktualisieren Sie es (<code>_autorun_taskset_prefix</code>) und starten Sie das Panel neu starten neu.
REPORTING-Abzeichen fehlt im Wasserfall	Das Reporting muss mit mindestens einem Slot eingeschaltet sein; das Abzeichen liest <code>/autorun-active.json</code> . Bauen Sie das Frontend neu, wenn <code>dist/index.html</code> älter als das Abzeichen ist.

KAPITEL 4

Thermal Guard

THERMAL_GUARD.md

Thermal Guard — Handbuch für Sysops

Schutz vor CPU-Überhitzung für PhantomSDR-Plus.

Ein WebSDR ist eine der wenigen Anwendungen, die jeden Kern dauerhaft auslasten — den ganzen Tag, für immer. Wenn ein Lüfter verstopft, eine Pumpe ausfällt oder der Raum im August heiß wird, sagt Ihnen das niemand: Der Server dekodiert einfach weiter, bis sich die CPU selbst gart. Der Thermal Guard ist der Teil, der das bemerkt und für Sie den Stecker zieht.

Er wird **inaktiv ausgeliefert**: Ab Werk schreibt er nur in ein Protokoll. Nichts ist geschützt, bis Sie einen Modus wählen. In diesem Handbuch geht es darum, diese Wahl zu treffen und damit zu leben.

Eilig? Führen Sie `python3 thermal_guard.py --once` aus, lesen Sie die ausgegebenen Schwellen und setzen Sie `thermal_mode` im Admin-Panel auf `stop+restart`. Das ist die ganze Arbeit. Alles Weitere unten ist das *Warum*.

Inhalt

1. Was er tut
2. Schnellstart
3. Wie er entscheidet
4. Die vier Modi — was jeder tut und was Sie tun müssen
5. Den Modus wählen
6. Konfigurationsreferenz
7. Betrieb ohne Admin-Panel
8. Die Throttle-Stufe ohne root aktivieren
9. Unbeaufsichtigter Betrieb — wenn niemand zusieht
10. Testen, bevor Sie ihm vertrauen
11. Das Protokoll lesen
12. Lockout und Ratenbegrenzung
13. Fehlersuche
14. Abschalten oder entfernen

1. Was er tut

Alle 2 Sekunden liest der Guard die Die-Temperatur Ihrer CPU und vergleicht sie mit vier Schwellen. Bleibt die Temperatur lange genug über einer davon, steigt er eine Leiter hinauf:

```

normal → warn → throttle → stop → (abkühlen) → restart
          Zeile ins   CPU-Takt   Stopp-   Server   Start-
          Protokoll   absenken   Skript   bleibt   Skript
                                aus

```

Zwei Entwurfsentscheidungen sorgen dafür, dass er auf *jeder* Maschine funktioniert — es lohnt sich, sie zu verstehen, bevor Sie sich auf ihn verlassen:

Er fragt nie, wie Sie Ihren Server betreiben. systemd, eine Watchdog-Schleife, cron, tmux, eine nackte Shell — der Guard weiß es nicht und es interessiert ihn nicht. Stattdessen hält er, solange die Maschine zu heiß ist, einen **Lockout**: Er wiederholt den Stopp bei *jedem* Takt. Startet Ihr Watchdog den Server neu, stoppt der Guard ihn innerhalb von zwei Sekunden wieder. Was auch immer den Server am Leben halten will, verliert — bis die CPU abgekühlt ist. Genau das macht ihn universell, und es ist gegen einen echten Watchdog getestet.

Er verdrahtet nie eine feste Temperatur. „Stopp bei 95 °C“ ist eine schlechte Regel: Auf einem Intel mit Tjmax 100 ist das eher normale Last, auf einem Ryzen liegt es *unter* dem Wert, auf dem Tctl unter Boost von Haus aus sitzt, und auf einem Raspberry Pi ist es unerreichbar, weil der SoC sich bei 80 selbst drosselt. Also liest der Guard den kritischen Grenzwert, den Ihr eigener Kernel veröffentlicht (`tempN_crit`), und rechnet davon zurück. Gleicher Code, überall sinnvolle Zahlen.

2. Schnellstart

1. Sehen Sie, wie Ihre Maschine für den Guard aussieht. Immer zuerst:

```
cd ~/PhantomSDR-Plus
python3 thermal_guard.py --once
```

```
sensor      : hwmon:coretemp
temperature: 65.0 C
crit        : 100.0 C
mode        : log
thresholds  : warn 88.0  throttle 92.0  stop 95.0  resume 75.0 (C)
cpufreq     : not writable – throttle stage disabled
```

Steht bei `sensor` ein `NONE`, hören Sie hier auf und lesen Sie die [Fehlersuche](#) — der Guard kann keine Maschine schützen, die er nicht messen kann, und er sagt Ihnen das, statt so zu tun als ob.

2. Lassen Sie ihn ein paar Tage im Modus `log`. Er läuft bereits im Admin-Panel. Lassen Sie ihn eine normale Woche beobachten — Sommernachmittage, einen großen Build, einen betriebsamen Contest-Samstag — und sehen Sie nach, ob er je ausgelöst hätte.

3. Scharfschalten. Admin-Panel → **Settings** → **THERMAL GUARD** → **MODE** → `stop+restart` → **SAVE THERMAL SETTINGS**. Wirkt nach etwa 2 Sekunden; kein Neustart nötig.

3. Wie er entscheidet

Der Sensor

Der Guard vertraut nur echten CPU-Die-Sensoren:

Quelle	Typische Hardware
<code>coretemp</code>	Intel
<code>k10temp</code> , <code>zenpower</code> , <code>zenpower3</code>	AMD
<code>cpu_thermal</code> , <code>soc_thermal</code>	Raspberry Pi, ARM-SBCs
Thermal Zone <code>x86_pkg_temp</code>	Rückfallebene auf x86

acpitz und unbeschriftete Thermal Zones sind **absichtlich ausgeschlossen**. Auf einem typischen Desktop meldet acpitz rund 28 °C, während das CPU-Package bei 59 °C liegt — ein Guard, der dem vertraut, würde schlicht nie auslösen. „Deaktiviert, kein vertrauenswürdiger Sensor“ zu melden ist ehrlicher, als Schutz vorzutäuschen.

psutil wird genutzt, falls es zufällig installiert ist, ist aber nie erforderlich. Der Guard kommt mit der reinen Standardbibliothek aus.

Die Schwellen

Jede ist ein Abstand unterhalb des kritischen Grenzwerts, den die Maschine selbst veröffentlicht:

Schwelle	Abstand unter crit	Bedeutung
warn	−12 °C	Etwas stimmt nicht, protokollieren
throttle	−8 °C	Durch Absenken des Takts abkühlen
stop	−5 °C	Zu nah am Limit — Server stoppen
resume	−25 °C	Wirklich wieder kühl, Rückkehr sicher

In der Praxis ergibt das:

Ihre CPU meldet	warn	throttle	stop	resume
Intel, crit 100 °C	88	92	95	75
AMD Ryzen, crit 95 °C	83	87	90	70
Raspberry Pi, crit 85 °C	73	77	80	60

Veröffentlicht die Maschine gar keinen kritischen Punkt, fällt der Guard auf einen konservativen festen Satz zurück und sagt das auch. Sie können jede Schwelle mit einem absoluten Wert überschreiben (siehe [Konfigurationsreferenz](#)), sollten es aber nicht müssen.

Haltezeiten — warum eine Spitze nie auslöst

Bei einem einzelnen heißen Messwert passiert nichts. Eine Schwelle muss **durchgehend** gehalten werden:

- **warn** und **throttle**: 30 s (`thermal_warn_sustain_s`)
- **stop**: 60 s (`thermal_sustain_s`)
- **resume**: 300 s unterhalb der Resume-Schwelle vor einem Neustart (`thermal_resume_s`)

Eine Compile-Spitze, ein Schub Dekodierarbeit, ein hochdrehender Lüfter — alles viel zu kurz. Ein einziger Wert unterhalb der Schwelle setzt die Uhr zurück. Zusätzlich verlangt der Guard nach dem Start 3 aufeinanderfolgende gültige Messwerte, bevor er überhaupt handelt, und verwirft jeden Wert außerhalb von 20–125 °C als Sensorfehler statt als Temperatur.

4. Die vier Modi

`thermal_mode` ist der eine Schalter, der bestimmt, wie weit der Guard die Leiter hinaufsteigen darf. Jeder Modus **umfasst alles, was die vorherigen tun**.

Modus	Warnt	Drosselt	Stoppt	Startet neu	Server kann ausfallen?
log	✓	—	—	—	Nein — rührt nichts an
throttle	✓	✓	—	—	Nein
stop	✓	✓	✓	—	Ja, bleibt aus bis Sie handeln
stop+restart	✓	✓	✓	✓	Ja, kommt selbst zurück

log — beobachten und melden (Auslieferungszustand)

Was er tut: alles, was die anderen Modi tun, nur ohne zu handeln. Er durchläuft die ganze Leiter und schreibt in `crash.log`, was er *getan hätte*. Er führt nie Ihr Stopp-Skript aus, rührt die CPU-Frequenz nicht an und unterbricht keinen Zuhörer.

Was Sie tun müssen: nichts — er läuft bereits. Öffnen Sie nach einer Woche den Reiter **CRASH** im Admin-Panel oder führen Sie aus:

```
grep THERMAL crash.log
```

- **Nichts da?** Ihre Kühlung ist in Ordnung und Sie können den Guard beruhigt scharfschalten.
- **STOP level reached ... NOT stopping?** Der Guard hätte Ihren Server heruntergefahren. Das ist ein echtes thermisches Problem, mit dem Sie bisher gefahren sind — beheben Sie die Kühlung *und* schalten Sie den Guard scharf.

Nutzen Sie ihn, wenn: Sie den Guard gerade installiert haben und noch nicht wissen, wie sich Ihre Maschine verhält. Das ist ein Erprobungsmodus, kein Ziel — ein Guard im Modus `log` schützt nichts.

throttle — CPU verlangsamen, Betrieb nie unterbrechen

Was er tut: An der Throttle-Schwelle senkt er die maximale CPU-Frequenz um 20 % und stellt sie wieder her, sobald die Temperatur unter die Warn-Linie zurückgekehrt ist. Ihr WebSDR bleibt durchgehend online. Zuhörer bemerken unter hoher Last vielleicht etwas schlechtere Leistung; die meisten bemerken gar nichts.

Steigt die Temperatur trotzdem über die Stopp-Schwelle, **protokolliert dieser Modus das nur und tut nichts** — er stoppt den Server nicht.

Was Sie tun müssen:

1. Dem Guard Schreibzugriff auf das CPU-Frequenzlimit geben — er braucht ihn, und standardmäßig hat ihn nur root. Führen Sie einmalig `./setup-cpufreq-perms.sh` aus; siehe [Abschnitt 8](#).
2. Mit `python3 thermal_guard.py --once` prüfen, dass dort `cpufreq : writable, throttle stage available` steht. Steht dort *not writable*, tut dieser Modus nichts, was `log` nicht auch täte — und sagt das jedes Mal im Protokoll.

Nutzen Sie ihn, wenn: Sie einem grenzwertigen Kühlproblem nachgehen und die Maschine verteidigen wollen, ohne je Zuhörer zu verlieren. **Beachten Sie: Das ist kein vollständiger Schutz** — reicht das Drosseln nicht, passiert nichts weiter. Als erster Schritt gut, als endgültige Antwort schlecht.

stop — Server herunterfahren und aus lassen

Was er tut: An der Stopp-Schwelle führt er Ihr Stopp-Skript aus und hält dann den **Lockout** — er stoppt den Server alle 2 Sekunden erneut, solange die Maschine zu heiß ist, damit ihn nichts hinter Ihrem Rücken wiederbelebt. War die CPU 5 Minuten lang unterhalb der Resume-Schwelle, endet der Lockout.

Der Server kommt nicht von selbst zurück. Sie starten ihn neu, wenn Sie überzeugt sind, dass die Ursache behoben ist.

Was Sie tun müssen:

1. Sicherstellen, dass der Guard weiß, wie er Ihren Server stoppt. Im Admin-Panel ist das das konfigurierte Stopp-Skript (`./stop-websdr.sh` bei den mitgelieferten Startskripten). Ist keines gesetzt, greift der Guard auf `SIGTERM` an den Prozessnamen zurück und nach 10 Sekunden Gnadenfrist auf `SIGKILL` — das funktioniert, aber ein richtiges Stopp-Skript ist sauberer.
2. Akzeptieren, dass **Ihr WebSDR offline bleibt, bis Sie es bemerken**. Sorgen Sie dafür, dass Sie es erfahren: eine Uptime-Überwachung, der CRASH-Reiter oder schlicht tägliches Nachsehen.

Nutzen Sie ihn, wenn: die Maschine wichtiger ist als der Dienst, Sie meist vor Ort sind, oder ein thermisches Ereignis ernst genug ist, dass Sie die Kiste ansehen wollen, bevor sie weiterläuft. Auch die richtige Wahl, wenn Sie einem automatischen Neustart auf Ihrer Hardware nicht trauen.

stop+restart — herunterfahren, abkühlen, selbst zurückkommen

Was er tut: alles wie `stop`, und zusätzlich: Hat die CPU 5 Minuten lang unter der Resume-Schwelle gelegen, führt er Ihr Start-Skript aus und das WebSDR kommt von selbst zurück.

Gegen Flattern schützt `thermal_max_stops_hour` (Standard 2). Nach zwei thermischen Stopps innerhalb einer Stunde **deaktiviert** der Guard den automatischen Neustart und lässt den Server aus, damit Sie nachsehen. Beachten Sie das wichtige Detail: Die Ratenbegrenzung schaltet den *Neustart* ab, nie den *Schutz* — der Lockout stoppt den Server weiterhin, solange er zu heiß ist, ganz gleich wie oft er schon ausgelöst hat.

Was Sie tun müssen:

1. Sicherstellen, dass **beide** Skripte konfiguriert sind und eigenständig funktionieren — das Start- ebenso wie das Stopp-Skript. Testen Sie sie von Hand: `./stop-websdr.sh`, dann `./start-rx888mk2.sh`.
2. Sonst nichts. Das ist der Modus zum Vergessen.

Nutzen Sie ihn, wenn: das WebSDR öffentlich, unbeaufsichtigt oder entfernt ist — also bei den meisten. Das ist der empfohlene Modus für eine normale Installation.

5. Den Modus wählen

Ihre Lage	Modus
Guard gerade installiert, Maschine noch unbekannt	eine Woche <code>log</code>
Öffentliches WebSDR, unbeaufsichtigt, soll sich selbst versorgen	stop+restart
Entfernter Standort, schwer erreichbar	stop+restart
Sie wollen die Maschine nach jedem thermischen Ereignis ansehen	<code>stop</code>
Hardware, der Sie einen automatischen Neustart nicht zutrauen	<code>stop</code>
Grenzwertige Kühlung, dürfen keine Zuhörer verlieren	<code>throttle</code> , danach weiter zu <code>stop+restart</code>
Maschine ohne vertrauenswürdigen CPU-Sensor	keiner davon funktioniert — erst den Sensor klären

Für die meisten Sysops lautet die Antwort `stop+restart`, und die ehrliche Zusammenfassung der anderen ist: `log` schützt nichts und `throttle` schützt ein wenig.

6. Konfigurationsreferenz

Alle Schlüssel liegen in `admin_config.json` neben `thermal_guard.py`, und jeder davon ist auch unter **Settings → THERMAL GUARD** editierbar. Der Guard **liest die Datei bei jedem Takt neu**, Änderungen wirken also innerhalb von ~2 Sekunden — ohne Neustart.

Schlüssel	Standard	Bedeutung
<code>thermal_enabled</code>	<code>true</code>	Hauptschalter. <code>false</code> deaktiviert den Guard vollständig.
<code>thermal_mode</code>	<code>"log"</code>	<code>log</code> <code>throttle</code> <code>stop</code> <code>stop+restart</code> — siehe Abschnitt 4 .
<code>thermal_warn</code>	<code>null</code>	Absolute Warn-Schwelle in °C. <code>null</code> = aus dem crit-Wert ableiten.
<code>thermal_throttle</code>	<code>null</code>	Absolute Throttle-Schwelle. <code>null</code> = abgeleitet.
<code>thermal_stop</code>	<code>null</code>	Absolute Stopp-Schwelle. <code>null</code> = abgeleitet.
<code>thermal_resume</code>	<code>null</code>	Absolute Resume-Schwelle. <code>null</code> = abgeleitet.
<code>thermal_sustain_s</code>	<code>60</code>	Sekunden, die die Stopp-Schwelle vor dem Stoppen gehalten werden muss. Minimum 5.
<code>thermal_warn_sustain_s</code>	<code>30</code>	Sekunden für die Warn- und die Throttle-Stufe. Minimum 5.
<code>thermal_resume_s</code>	<code>300</code>	Sekunden unterhalb der Resume-Schwelle, bis der Lockout endet. Minimum 30.
<code>thermal_max_stops_hour</code>	<code>2</code>	Thermische Stopps pro Stunde, nach denen der automatische Neustart abgeschaltet wird. Der Schutz läuft weiter.
<code>thermal_test_temp</code>	<code>null</code>	Tut so, als hätte die CPU diese Temperatur. Nur zum Testen — siehe Abschnitt 10 .

Der eigenständige Guard nutzt zusätzlich `start_script`, `stop_script`, `sdr_process_name` und `sdr_base_dir` aus derselben Datei.

Eine Schwelle überschreiben. Tun Sie das nur mit gutem Grund. Ein häufiger: Ihre Maschine läuft bei Builds legitim heiß und Sie möchten nicht, dass das zählt. `thermal_stop` anzuheben kauft Ihnen Luft auf Kosten des Abstands zum kritischen Punkt — setzen Sie ihn nie über Ihren crit-Wert, sonst erreicht die CPU zuerst ihre eigene Notabschaltung und der Guard ist reine Dekoration.

7. Betrieb ohne Admin-Panel

Viele Sysops installieren das Panel nie. Der Guard läuft problemlos für sich allein — dieselbe Datei, dasselbe Verhalten.

Zuerst im Vordergrund ausprobieren:

```
cd ~/PhantomSDR-Plus
python3 thermal_guard.py --once          # was passierte hier?
python3 thermal_guard.py --mode log      # live zusehen, Strg-C beendet
```

`--mode` überschreibt `thermal_mode` von der Kommandozeile, Sie müssen also nie JSON von Hand schreiben:

```
python3 thermal_guard.py --mode stop+restart
python3 thermal_guard.py --config /etc/phantomsdr/thermal.json
```

Dann dauerhaft einrichten mit der im Projekt mitgelieferten Beispiel-Unit:

```
sudo cp thermal-guard.service /etc/systemd/system/
sudo nano /etc/systemd/system/thermal-guard.service # User= und Pfade setzen
sudo systemctl daemon-reload
sudo systemctl enable --now thermal-guard
systemctl status thermal-guard
```

Lesen Sie die Kommentare am Anfang von `thermal-guard.service`, bevor Sie sie aktivieren. Zwei Punkte sind am wichtigsten:

- **User= muss ein Konto sein, das Ihre Start- und Stopp-Skripte wirklich ausführen kann.** Den Guard als root laufen zu lassen ist meist die falsche Antwort: Er würde diese Skripte dann ebenfalls als root ausführen.
- Verweisen Sie ihn entweder auf eine `admin_config.json` mit `start_script` und `stop_script`, oder geben Sie `--mode` in der `ExecStart`-Zeile mit. Eine minimale Konfiguration genügt:

```
{
  "sdr_base_dir": "/home/ihrbenutzer/PhantomSDR-Plus",
  "sdr_process_name": "spectrumserver",
  "start_script": "start-rx888mk2.sh",
  "stop_script": "stop-websdr.sh",
  "thermal_mode": "stop+restart"
}
```

Alles, was der Guard tut, landet weiterhin in `crash.log` neben dieser Konfigurationsdatei sowie im systemd-Journal (`journalctl -u thermal-guard`).

8. Die Throttle-Stufe ohne root aktivieren

Standardmäßig meldet `python3 thermal_guard.py --once`:

```
cpufreq : not writable – throttle stage disabled
```

Das ist normal und kein Fehler. Der Kernel legt

```
/sys/devices/system/cpu/cpuN/cpufreq/scaling_max_freq
```

als `root:root` mit Modus `0644` an. Ein Admin-Panel, das als gewöhnlicher Benutzer läuft, kann dort nicht schreiben — der Guard überspringt die Throttle-Stufe und sagt das im Protokoll, statt stillschweigend zu versagen.

Die falsche Lösung ist, den Guard als root laufen zu lassen. Er führt auch Ihre Start- und Stopp-Skripte aus, und die sollen unprivilegiert bleiben.

Die richtige Lösung ist, den Schreibzugriff auf genau diese Dateien einer eigenen Gruppe zu geben. Ein Skript erledigt das:

```
cd ~/PhantomSDR-Plus
./setup-cpufreq-perms.sh
```

Es fragt einmal nach Ihrem Passwort — und nur einmal, bei der Einrichtung — und dann:

1. legt es eine Systemgruppe `cpufreq` an und nimmt Sie hinein;

2. installiert `/etc/tmpfiles.d/99-phantomsdr-cpufreq.conf`, damit Gruppenzugehörigkeit und Modus 0664 **bei jedem Systemstart** neu gesetzt werden — sysfs-Rechte überstehen einen Neustart nicht von selbst;
3. wendet die Änderung sofort an, damit Sie ohne Neustart testen können.

Danach, in dieser Reihenfolge:

```
# 1. ab- und wieder anmelden (oder neu starten) – eine neue Gruppe
# erreicht Ihre Prozesse nur über eine frische Anmeldung
id | grep cpufreq

# 2. Panel neu starten; der Guard prüft den Schreibzugriff einmal beim Start
sudo systemctl restart phantomsdr-admin
# (so ist kein Abmelden nötig – systemd baut die Gruppenliste bei jedem
# Start neu auf. Nur wenn die Units NICHT installiert sind:
# ./manage_admin.sh stop && ./manage_admin.sh start – niemals beides, sie streiten um Port
# 3000)

# 3. bestätigen
python3 thermal_guard.py --once
# cpufreq : writable, throttle stage available
```

Vollständig rückgängig machen: `sudo ./setup-cpufreq-perms.sh --revoke`.

Hinweise

- Die Throttle-Stufe *handelt* weiterhin nur, wenn `thermal_mode` auf `throttle` oder höher steht. Die Berechtigung allein ändert nichts.
- Der Guard senkt das *aktuell* geltende Limit um 20 % und stellt genau diesen Wert wieder her. Wenn Sie Ihre CPU bereits begrenzen (etwa über `MAX_SPEED` in `/etc/init.d/cpufrequtils`), wird Ihre Begrenzung wiederhergestellt — der Guard gibt Ihnen nicht klammheimlich einen höheren Takt zurück, als Sie eingestellt haben.
- Manche Maschinen legen überhaupt keinen `cpufreq`-Treiber offen — viele VPS, die meisten Container. Das Skript erkennt das und ändert nichts. Verwenden Sie dort `stop` oder `stop+restart`; die hängen nicht von `cpufreq` ab.

9. Unbeaufsichtigter Betrieb

Die häufigste Frage zum Guard: *Was passiert, wenn niemand zusieht?* Die kurze Antwort: Genau dafür wurde er gebaut.

Er braucht kein offenes Browserfenster. Der Guard ist ein Hintergrund-Thread in `admin_server.py`, getaktet vom Graphen-Sampler alle 2 Sekunden. Er läuft, ob jemand im Panel angemeldet ist oder nicht, ob ein Browser offen ist oder nicht, ob Sie wach sind oder nicht. Im eigenständigen Betrieb ist er ein systemd-Dienst und damit noch unabhängiger. Die Dashboard-Kachel ist ein *Fenster* auf den Guard, nicht der Guard selbst.

Er handelt ohne zu fragen. Es gibt keinen Bestätigungsdialog und keine Benachrichtigung, auf die gewartet wird. An der Stopp-Schwelle stoppt er den Server, Punkt. Genau darum geht es.

Aber nur, wenn Sie es erlaubt haben. Im Modus `log` protokolliert er getreulich, wie sich eine Maschine gart, und tut nichts dagegen. Wenn Sie diesen Abschnitt lesen, weil „*ich bin nicht da, um zuzusehen*“, ist `stop+restart` der Modus, den Sie wollen — nur er schützt die Maschine und bringt den Dienst ohne Sie zurück.

Er hilft nur, solange das Panel lebt. Der Wächter ist Teil von `admin_server.py`: Ein Panel, das um 02:00 Uhr gestorben ist, nimmt den Schutz mit, und nach einem Neustart bleibt die Maschine

ungeschützt, bis Sie sich anmelden und es erneut starten. Wenn niemand zusieht, lassen Sie systemd zusehen: Das Repository liefert `phantomsdr-admin.service` und `phantomsdr-proxy.service` mit, die beim Booten starten und das Panel innerhalb von fünf Sekunden nach einem Absturz neu starten. Siehe [ADMIN_PANEL_SETUP.md](#). Für einen unbeaufsichtigten Empfänger wird das **dringend empfohlen** — `stop+restart` scharfzuschalten schützt Sie sonst nur bis zum nächsten Neustart.

Wie Sie es hinterher erfahren. Grob nach Nützlichkeit:

Wo	Was Sie bekommen
<code>crash.log</code> / Reiter CRASH	Jede Aktion mit Zeitstempel. Die dauerhafte Aufzeichnung.
Dashboard-Kachel THERMAL GUARD	Live-Zustand: Temperatur, Stufe, Lockout, Stopps der letzten Stunde.
Seite Graphs (CPU-Temperatur, 24 h)	Der Verlauf — wie schnell es stieg, wie lange das Abkühlen dauerte.
<code>journalctl -u thermal-guard</code>	Nur im eigenständigen Betrieb.

Eine sinnvolle Routine für einen unbeaufsichtigten Standort: `stop+restart` scharf, und `grep THERMAL crash.log`, wann immer Sie sich ohnehin anmelden. Ist es leer, gibt es nichts zu wissen.

Ein durchgespieltes Beispiel. Lüfterausfall um 03:00 auf einer Maschine im Modus `stop+restart`:

```
03:14 Temperatur überschreitet 88 °C, hält 30 s      → warn protokolliert
03:16 überschreitet 92 °C, hält 30 s                → throttle angewandt (falls erlaubt)
03:19 überschreitet 95 °C, hält 60 s                → STOP – Stopp-Skript, Lockout an
03:19 Watchdog startet den Server neu              → Guard stoppt erneut, protokolliert
03:26 CPU seit 5 Minuten unter 75 °C                → Lockout endet, Start-Skript läuft
03:41 überhitzt ein zweites Mal und stoppt          → Ratengrenze erreicht, kein Neustart mehr
                                                    Server bleibt aus, Schutz bleibt an
```

Sie wachen auf mit einem WebSDR, das aus ist, einer `crash.log`, die genau sagt warum, und — vor allem — einer CPU, die ihrem kritischen Punkt nie nahe gekommen ist.

10. Testen, bevor Sie ihm vertrauen

Warten Sie nicht auf eine echte Hitzewelle, um herauszufinden, ob Ihr Stopp-Skript funktioniert.

`thermal_test_temp` lässt den Guard glauben, die CPU habe eine von Ihnen gewählte Temperatur. Alles andere verhält sich völlig normal — die Haltezeiten, die Leiter, der Lockout, Ihre echten Skripte.

1. Admin-Panel → **Settings** → **THERMAL GUARD** → **TEST TEMPERATURE**.
2. Einen Wert über Ihrer Stopp-Schwelle eintragen (etwa 97 auf einem Intel).
3. Speichern und die Dashboard-Kachel sowie den CRASH-Reiter beobachten.
4. **Feld danach leeren.** Eine stehengebliebene Testtemperatur bedeutet, dass der Guard nicht mehr den echten Sensor liest.

Im Modus `log` zeigt Ihnen das die ganze Leiter, ohne dass etwas auf dem Spiel steht — der sichere erste Test und der, den man vor jedem Scharfschalten macht.

Im Modus `stop+restart` ist das ein **Live-Test**: Ihr Server wird wirklich stoppen und etwa fünf Minuten nach dem Leeren des Feldes wirklich zurückkommen. Machen Sie das, wenn niemand zuhört. Es lohnt sich einmal, denn nur so wissen Sie, dass Ihre Stopp- und Start-Skripte auch funktionieren, wenn etwas anderes als Sie sie aufruft.

Entsprechungen im eigenständigen Betrieb:

```
python3 thermal_guard.py --once          # nur Schwellen, keine Aktion
python3 thermal_guard.py --mode log      # die Leiter live beobachten
```

11. Das Protokoll lesen

Alles wird an `crash.log` neben der Konfigurationsdatei angehängt, ein Ereignis pro Zeile, jeweils mit `[THERMAL]` davor:

```
grep THERMAL crash.log
```

Die Zeilen sind bewusst einzeilig und grep-freundlich. Was Sie sehen werden:

Zeile	Bedeutung
warn: 88.4C (warn=88.0 crit=100) sustained 30s	Erste Sprosse. Es wurde nichts getan.
throttle: 92.1C sustained 30s – cpufreq max lowered 20%	CPU-Takt abgesenkt.
throttle level reached: ... (no cpufreq write access – stage skipped)	Siehe Abschnitt 8 .
STOP level reached: ... mode is 'log', NOT stopping	Er hätte den Server gestoppt. Schalten Sie ihn scharf.
STOPPED server: 95.2C sustained 60s (stop=95.0 crit=100) – ran stop-websdr.sh	Der Ernstfall.
lockout: process reappeared at 96.0C – re-stopped (...) [12 time(s) so far]	Etwas startet Ihren Server neu; der Guard gewinnt. Auf eine Zeile pro Minute begrenzt.
rate limit: 2 thermal stops within the hour – automatic restart is now DISABLED	Bleibt aus, damit Sie nachsehen. Der Schutz läuft weiter.
recovered: 54.0C held below 75.0 – lockout cleared	Wieder kühl.
auto-restart: ran start-rx888mk2.sh	Wieder online.
back to normal: 59.0C (warn=88.0)	Von der Leiter gestiegen, ohne je zu stoppen.

12. Lockout und Ratenbegrenzung

Diese beiden Verhaltensweisen überraschen am ehesten, deshalb hier deutlich:

Der Lockout ist das, was den Guard gerade gegen *Ihre* Einrichtung wirksam macht. Wenn er den Server stoppt, setzt er nicht einfach einen Stopp ab und hofft. Er markiert sich als gesperrt und prüft bei jedem 2-Sekunden-Takt, solange die Temperatur über der Resume-Schwelle liegt, ob der Serverprozess wieder da ist — und stoppt ihn dann erneut. Ihr Watchdog, `Restart=always` in `systemd`, ein cron-Job, ein ungeduldiger Sysop: Sie alle verlieren diesen Streit, bis die CPU abkühlt. Er endet automatisch, sobald die Temperatur `thermal_resume_s` lang (Standard 5 Minuten) unter der Resume-Schwelle geblieben ist.

Die Schaltfläche CLEAR LOCKOUT auf der Dashboard-Kachel beendet den Lockout *und* setzt den Stopps-pro-Stunde-Zähler zurück — für den Fall, dass Sie die Kühlung repariert haben und nicht warten wollen. Sie **entschärft den Guard nicht**: Ist die Maschine noch heiß, stoppt die nächste Prüfung den Server sofort wieder. Das ist so beabsichtigt.

Die Ratenbegrenzung (`thermal_max_stops_hour`, Standard 2) verhindert, dass eine defekte Maschine die ganze Nacht auf und ab fährt. Ist die Grenze erreicht, **deaktiviert** der Guard den automatischen Neustart und lässt den Server aus. Lesen Sie das genau: Er deaktiviert den Neustart, nicht den Schutz. Der Lockout stoppt den Server weiterhin, solange er zu heiß ist, ganz gleich wie oft er schon ausgelöst hat. Ein Guard, der ausgerechnet dann verstummt, wenn es der Maschine am schlechtesten geht, wäre schlimmer als gar kein Guard.

13. Fehlersuche

sensor : NONE – no trusted CPU sensor on this machine Der Guard fand keinen zugelassenen Sensor und weigert sich zu raten. Probieren Sie `sensors` (aus `lm-sensors`; `sudo apt install lm-sensors && sudo sensors-detect`). In einer VM oder einem Container gibt es oft tatsächlich keinen Die-Sensor — der Guard kann diese Maschine nicht schützen und meldet sich korrekt als deaktiviert, statt so zu tun als ob.

cpufreq : not writable – throttle stage disabled Zu erwarten, solange Sie `./setup-cpufreq-perms.sh` nicht ausgeführt haben — siehe [Abschnitt 8](#). Betrifft nur die Throttle-Stufe; `stop` und `stop+restart` sind davon unberührt.

setup-cpufreq-perms.sh ausgeführt, meldet trotzdem „not writable“ Zwei wahrscheinliche Ursachen: Sie haben sich seitdem nicht ab- und wieder angemeldet (eine neue Gruppe erreicht Prozesse nur über eine frische Anmeldung — mit `id` prüfen), oder das Panel wurde seitdem nicht neu gestartet (der Guard prüft den Schreibzugriff einmal beim Start). Beides nachholen, in dieser Reihenfolge.

Die Schwellen wirken falsch für meine CPU Prüfen Sie, was die Maschine veröffentlicht: `cat /sys/class/hwmon/hwmon*/temp*_crit`. Ist Ihr `crit`-Wert ungewöhnlich oder fehlt er, setzen Sie `thermal_stop` und Verwandte auf absolute Werte.

Er hat meinen Server gestoppt, obwohl es meiner Meinung nach nicht heiß war Prüfen Sie, ob `thermal_test_temp` noch von einem Test gesetzt ist. Das ist mit Abstand die häufigste Ursache.

Er löst nie aus, obwohl die Maschine heiß wird Prüfen Sie, dass der Modus nicht `log` ist, dass `thermal_enabled` auf `true` steht, und vergleichen Sie die gesehene Temperatur mit den Schwellen aus `--once`. Denken Sie daran, dass die Schwelle die Haltezeit über *durchgehend* überschritten sein muss.

Der Server startet während eines thermischen Ereignisses ständig neu Das ist der Lockout bei der Arbeit, und die `lockout: -`Zeilen sind seine Erfolgsmeldung — Ihr Supervisor versucht es weiter, der Guard macht es weiter rückgängig. Nichts zu reparieren.

Das Panel läuft nicht — läuft dann der Guard? Nein — im Panel-Betrieb lebt der Guard in `admin_server.py`. Wollen Sie Schutz unabhängig vom Panel, nutzen Sie den eigenständigen Dienst aus [Abschnitt 7](#).

14. Abschalten oder entfernen

- **Pausieren:** `thermal_mode` auf `log` setzen. Er beobachtet und meldet weiter, handelt aber nie.
- **Vollständig deaktivieren:** `thermal_enabled` auf `false` setzen.
- **Eigenständig:** `sudo systemctl disable --now thermal-guard`.
- **cpufreq-Rechte zurücknehmen:** `sudo ./setup-cpufreq-perms.sh --revoke`.

`thermal_guard.py` zu löschen ist ebenfalls unbedenklich — `admin_server.py` importiert die Datei in einem `try` und meldet den Guard einfach als nicht verfügbar, wenn sie fehlt.

Siehe auch

- [Admin-Panel-Einrichtung](#) — der Reiter Thermal Guard im Zusammenhang
- [Installation](#) — thermischer Schutz bei einer Neuinstallation
- [Projektstruktur](#) — wo `thermal_guard.py` liegt

KAPITEL 5

Verbindungslimits

CONNECTION_LIMITS.md

Verbindungslimits — Sysop-Handbuch

Einen öffentlichen Empfänger vor Verbindungsfluten schützen.

Ein WebSDR ist ein öffentlicher Dienst, der jedem, der danach fragt, einen fortlaufenden Strom aus Audio und Spektrum liefert. Genau das ist sein Zweck — und genau darin liegt das Problem: für den Server sieht ein normaler Hörer nicht anders aus als jemand, der Sitzungen in einer Schleife öffnet. Am 16. September 2026 traf es diesen Empfänger genau so — eine einzige Adresse öffnete 134 Hörsitzungen innerhalb einer Minute und hielt 181 davon gleichzeitig offen, während das echte Publikum der Station bei etwa einem Dutzend lag.

Dieses Handbuch beschreibt die drei Schichten, die das verhindern, was jede von ihnen leisten kann und was nicht, und woran Sie erkennen, ob Ihre vernünftig eingestellt sind.

Wenig Zeit? Die Beispiel-`config.toml` bringt bereits sinnvolle Werte mit, eine frische Installation ist also ohne Ihr Zutun geschützt. Wenn Ihre eigene `.toml` älter ist und außer `audio/waterfall/events` keine `[limits]`-Schlüssel enthält, sind die Limits pro Adresse schlicht **aus** — kopieren Sie den Block aus dem Beispiel, um sie einzuschalten. Der eine Befehl, den Sie danach kennen sollten, ist `grep Refused spectrumserver.log`: er sagt Ihnen, ob jemand abgewiesen wird.

Inhalt

1. Wogegen Sie sich tatsächlich verteidigen
2. Die drei Schichten
3. Limits pro Adresse — die Hörer-Politik
4. Untätige Verbindungen — der stille Angriff
5. Der Kernel-Schutz — `setup-firewall.sh`
6. Konfigurationsreferenz
7. Was ein abgewiesener Hörer sieht
8. Logs und Zähler lesen
9. Ihre Zahlen wählen
10. Fallen, die man kennen sollte
11. Alles wieder abschalten

1. Wogegen Sie sich tatsächlich verteidigen

Genauigkeit lohnt sich hier, denn das Wort „DDoS“ deckt zwei sehr verschiedene Dinge ab, und nur eines davon lässt sich auf der eigenen Maschine lösen.

Flut aus einer Quelle. Ein Rechner, oder eine Handvoll, die so schnell Verbindungen öffnen wie sie können. Das ist hier passiert, und es ist vollständig behebbar: die Verbindungen kommen von einer Adresse, die man zählen kann, und Zählen genügt.

Ein echter verteilter Angriff. Tausende Rechner, oft mit gefälschten Absenderadressen, die Ihre Anbindung füllen. Wenn diese Pakete Ihre Netzwerkkarte erreichen, ist die Bandbreite bereits verbraucht. **Nichts aus diesem Handbuch hilft dagegen**, und nichts, was Sie auf dem Server installieren, kann es — die einzige Antwort ist ein Dienst oberhalb von Ihnen, was ein anderes Thema ist und eine echte Domain sowie einen Anbieter wie Cloudflare vor dem Empfänger voraussetzt.

Alles Folgende betrifft den ersten Fall. Das ist keine Einschränkung, für die man sich entschuldigen müsste: der erste Fall ist das, was Amateurstationen tatsächlich trifft.

2. Die drei Schichten



Sie sind bewusst nicht redundant. Die Kernel-Schicht ist schnell, aber dumm — sie zählt Pakete von einer Adresse und weiß nichts über Hörer. Die Server-Schicht ist präzise, sieht eine Verbindung aber erst, wenn sie angenommen und ausgewertet wurde, was etwas kostet. Die Browser-Schicht existiert, damit ein abgewiesener Hörer erfährt, was passiert ist, statt auf eine Seite zu starren, die nie lädt.

3. Limits pro Adresse

Was als ein Hörer zählt

Ein Hörer ist **eine /audio-Verbindung** — oder, wo `[kiwi_emulation]` eingeschaltet ist, **eine Kiwi-Sound-Verbindung**, die genau gleich zählt. Das ist dieselbe Einheit, die die Benutzerzahl, `users.json` und die Beschriftungen auf dem Wasserfall bereits verwenden, ein Kiwi-Client verbraucht also einen Platz und erscheint in der Benutzerliste wie ein Browser.

Das ist wichtig, weil ein einziger Browser-Tab auf dem Desktop **vier** Verbindungen zu Ihrem Server öffnet:

Verbindung	Zweck
/audio	der demodulierte Audiostrom — das ist der Hörer
/waterfall	die Spektrumanzeige
/events	Benutzerliste, Frequenzen der anderen Hörer
/chat	das Chat-Fenster

Ein Telefon auf /mobile öffnet zwei (/audio und /chat), weil es keinen Wasserfall hat. Ein Limit, das rohe Verbindungen zählte, würde also alle abweisen: drei Tabs sind zwölf Verbindungen.

Die drei Schlüssel

```
[limits]
per_ip=3          # gleichzeitige Hörer von einer Adresse
per_ip_sockets=0  # gleichzeitige Verbindungen jeder Art; 0 ergibt per_ip * 4 + 4
per_ip_rate=40    # neue Verbindungen pro Minute von einer Adresse
per_ip_ban_s=600  # wie lange eine Adresse abgewiesen wird, die die Rate reißt
```

per_ip ist die Kennzahl, über die Sie nachdenken werden. Drei ist ein guter Anfang.

per_ip_sockets gibt es, weil /audio nicht das Einzige ist, was sich zu fluten lohnt. /waterfall ist der schwerste Strom, den Ihr Server erzeugt, und ein Limit, das nur Hörer zählt, ließe sich schlicht umgehen, indem man stattdessen Wasserfall-Verbindungen öffnet. Auf 0 belassen wird daraus $\text{per_ip} * 4 + 4$, was per_ip vollen Desktop-Tabs entspricht, mit etwas Luft für Verbindungen, die der Kernel nach einem Neuladen noch nicht abgeräumt hat.

per_ip_rate ist das, was eine Flut tatsächlich beendet. Ein Limit allein tut das nicht: eine Verbindung abzuweisen ist billig, aber nicht umsonst, und ein Angreifer kann so schnell neu verbinden, wie die Absagen zurückkommen, und dabei Ihre CPU mit Auf- und Abbau verheizen. Das Ratenlimit gibt ihm eine Sperre statt einer Antwort. Denken Sie daran, dass ein Tab vier kostet: 40/Minute sind also etwa zehn Seitenaufrufe pro Minute von einer Adresse.

Eine bewusste Reihenfolge. Die Rate wird belastet, *bevor* das Gleichzeitigkeitslimit geprüft wird. Das wirkt verkehrt herum, bis man an eine Adresse denkt, die bereits an ihrem Limit sitzt und in einer Schleife neu verbindet: würde ihr jedes Mal die billige Absage „du bist am Limit“ antworten, sammeln sich nie Ratenverstöße an, und die Sperre, die die Schleife beendet, käme nie zustande.

Wer ausgenommen ist

Loopback wird nie limitiert. Der Autorun-Tab für die Spot-Meldungen, das Admin-Panel und ein auf dem Serverrechner selbst geöffneter Browser kommen alle von 127.0.0.1, und ließe man sie Plätze verbrauchen, würde Ihre eigene Station Ihre Hörer aussperren.

Das deckt auch proxy.py ab: Verbindungen über den Proxy erreichen spectrumserver von Loopback aus, und die echte Client-Adresse kommt im Header X-Forwarded-For an — und genau darauf sind die Limits geschlüsselt.

Seit v5.0.0 wird dieser Header **nur noch von einem Loopback-Partner** geglaubt — einem Proxy auf demselben Rechner. Von jedem anderen wird er ignoriert und die TCP-Adresse zählt, denn sonst könnte ein direkter Besucher X-Forwarded-For: 127.0.0.1 schreiben und als Loopback an jedem Limit dieser Seite vorbeigehen. proxy.py verwirft außerdem jedes X-Forwarded-For oder X-Real-IP, das ein Client mitschickt, bevor es seine eigenen anfügt, und beantwortet den Sysop-Kick (/~~kick) nur für Clients auf demselben Rechner. Eine Station mit mehreren Empfängern hinter dem Proxy beschreibt **Mehrere Empfänger**; die Limits gelten dort je Empfänger.

4. Untätige Verbindungen

Es gibt einen Angriff, den nichts von alledem sehen kann.

Eine TCP-Verbindung öffnen. Nichts senden. Sie offen halten.

So ein Gegenüber schließt nie eine Anfrage ab, erreicht also nie den Code, in dem die Limits pro Adresse leben — es gibt keinen Hörer, kein Client-Objekt, nichts, was einen Namen zum Zählen hätte. Ihn kostet es ein paar Bytes, Sie einen Dateideskriptor und einen Platz, solange sie gehalten wird.

```
[limits]
idle_per_ip=8      # untätige Verbindungen (vor der Anfrage) pro Adresse
idle_total=512     # Obergrenze über alle Adressen hinweg
```

Das Limit liegt auf dem **Wie viele**, nicht auf dem **Wie lange**, und diese Unterscheidung ist der ganze Entwurf. **Es gibt keine Frist für eine untätige Verbindung, und das mit Absicht.** Eine Frist scheint die naheliegende Antwort und ist eine Falle (siehe §10): es gibt legitime Gegenstellen, die sich verbinden und dann lange schweigen, bevor sie sprechen. Was sie von einem Angriff unterscheidet, ist die Menge — eine legitime Gegenstelle hält eine solche Verbindung, ein Angreifer Tausende.

Eine untätige Verbindung lebt also, bis sie eine Anfrage stellt oder stirbt, und ihr Platz wird in beiden Fällen freigegeben.

`idle_total` begrenzt eine Flut, die über viele Quelladressen verteilt ist. Ist die Obergrenze erreicht, werden **neue** untätige Verbindungen abgewiesen, statt bestehende zu verdrängen — eine legitime, langlebige Verbindung ist also nie das, was Platz machen muss.

Beides greift in dem Moment, in dem der Socket angenommen wird, anhand der Adresse der Gegenstelle — es gibt noch keine Anfrage, also auch kein X-Forwarded-For. Das ist in Ordnung: Loopback wird nicht gezählt, was alles über `proxy.py` abdeckt, und eine direkte Flut bringt ihre eigene Adresse mit.

5. Der Kernel-Schutz

`setup-firewall.sh` installiert eine nftables-Tabelle, die Last abfängt, bevor `spectrumserver` auch nur ein Byte liest. Sie ist bewusst grob, und ihre Zahlen liegen deutlich über denen in `config.toml`, damit ein normaler Hörer nie von beiden erwischt werden kann.

```
./setup-firewall.sh --show      # Regeln ausgeben, nichts ändern
sudo ./setup-firewall.sh --check # gegen Ihren Kernel prüfen
sudo ./setup-firewall.sh --apply # laden, mit 60-Sekunden-Rückfall
sudo ./setup-firewall.sh --persist # beim Booten neu laden
sudo ./setup-firewall.sh --status # die Paketzähler je Regel
sudo ./setup-firewall.sh --remove # alles rückgängig machen
```

Vier Dinge sind abgedeckt: eine Obergrenze gleichzeitiger Verbindungen je Quelladresse auf Ihren Proxy- und Empfängerports, eine Rate neuer Verbindungen je Quelle, eine Bremse gegen SSH-Bruteforce und Windows-Dateifreigabe, die für alles außerhalb privater Adressbereiche geschlossen wird — `smbd` lauscht auf vielen Maschinen auf `0.0.0.0`, und ob das von außen erreichbar ist, hängt an einem Router, den das Skript nicht sehen kann.

Die Ports werden aus `admin_config.json` gelesen, derselben Datei, die `proxy.py` verwendet, es folgt also Ihrer Installation.

Warum es Sie nicht aussperren kann

Zwei Eigenschaften, beide beabsichtigt:

- Die Policy der Tabelle ist **accept**, und sie verwirft nur ausdrücklich benannte Muster. Eine Regel, die nicht greift, lässt das Paket zu allem durch, was Sie sonst konfiguriert haben. Sie kann die Maschine nicht unerreichbar machen.
- **Bestehende Verbindungen werden in der allerersten Regel angenommen.** Die SSH-Sitzung, in der Sie tippen, ist von allem Folgenden nie betroffen.

Darüber hinaus bewaffnet `--apply` einen automatischen Rückfall: Regeln laden, dann innerhalb von 60 Sekunden bestätigen, sonst werden sie wieder entfernt. Sagen Sie nichts, schließen Sie das Terminal, verlieren Sie die Verbindung — das Regelwerk verschwindet von selbst.

Nutzen Sie dieses Fenster richtig. Prüfen Sie **von einem anderen Gerät, außerhalb Ihres eigenen Netzes**, dass der Empfänger noch lädt und dass sich eine *neue* SSH-Sitzung öffnen lässt. Der Test mit der Sitzung, die Sie schon haben, beweist nichts, denn sie wurde von Regel eins angenommen.

6. Konfigurationsreferenz

Jeder Schlüssel unten steht unter `[limits]` in Ihrer `.toml`-Datei. **Alle sind standardmäßig aus oder großzügig**, eine Konfiguration, die sie nicht erwähnt, verhält sich also genau wie bisher.

Schlüssel	Vorgabe	Bedeutung
<code>per_ip</code>	0 (aus)	Gleichzeitige Hörer pro Adresse
<code>per_ip_sockets</code>	$0 \rightarrow \text{per_ip} * 4 + 4$	Gleichzeitige Verbindungen jeder Art pro Adresse
<code>per_ip_rate</code>	0 (aus)	Neue Verbindungen pro Minute pro Adresse
<code>per_ip_ban_s</code>	600	Sekunden Abweisung nach Reißen der Rate
<code>idle_per_ip</code>	8	Untätige Verbindungen vor der Anfrage, pro Adresse
<code>idle_total</code>	512	Untätige Verbindungen über alle Adressen

Ein Schlüssel auf 0 schaltet die jeweilige Prüfung ab.

Die drei, die keine Limits sind

In `[limits]` stehen außerdem `audio`, `waterfall` und `events`, von upstream PhantomSDR übernommen, und **keines der drei wird durchgesetzt**. `waterfall` und `events` werden von keinem Code gelesen. `audio` wird an genau einer Stelle gelesen: es geht als `max_users` in die Registrierungs-JSON an die Verzeichnisse aus `[websdr] register_urls`, ist also die Zahl, die `sdr-list.xyz` und die anderen anzeigen, und tut überhaupt nichts, wenn `register_online=false` gilt. Wenn Sie eine Obergrenze für Hörer wollen, ist `per_ip` weiter oben das, was wirkt.

Noch einer, unter `[server]`

```
[server]
min_client_version=0
```

Kein Limit, sondern eine Abweisung — und weil sie denselben Schließcode verwendet, gehört sie hierher. Seiten melden den Stand, aus dem sie geladen wurden, als `/audio?v=N`, und alles unterhalb von `min_client_version` wird mit „*diese Seite ist veraltet — bitte neu laden*“ abgewiesen.

Es gibt das, weil der Server JavaScript, das bereits im Browser eines Nutzers läuft, nicht mehr erreichen kann. Ein Tab behält den Code, den er geladen hat, bis jemand neu lädt; eine Änderung an der Seite erreicht also nur die Hörer, die zufällig neu laden — und eine Station, die gerade das Sitzungsverhalten geändert hat, braucht die alten Seiten unter Umständen jetzt weg und nicht irgendwann.

Geprüft wird nur `/audio`, denn dort lebt eine Sitzung, und Loopback sowie die Kiwi-Pfade sind ausgenommen, damit der Autorun-Tap und Kiwi-Clients unberührt bleiben. `0` schaltet es ab und ist die Vorgabe. Es eingeschaltet zu lassen, nachdem die alten Seiten abgeflossen sind, ist unbedenklich: Ein aktueller Diversity-Client, der auf einem blanken `/audio` abgewiesen wird, versucht es erneut mit der Kennung. Abgewiesen werden also nur noch Diversity-Clients mit einem älteren PhantomSDR-Plus-Stand und Fremdwerkzeuge, die `/audio` ohne `?v=` öffnen. Setzen Sie es zurück auf `0`, wenn eine andere Station meldet, dass sie Sie nicht mehr als Diversity-Partner nutzen kann.

7. Was ein abgewiesener Hörer sieht

Nichts verbindet sich je neu

Fangen Sie hier an, denn davon hängt alles Weitere ab: **eine abgebrochene `/audio`-Verbindung beendet die Sitzung**. Es gibt kein automatisches Wiederverbinden, und das mit Absicht.

Wiederverbinden ist der richtige Reflex für eine Chat-Anwendung und der falsche für einen Empfänger. `/waterfall` und `/events` kamen damit nie zurück, eine erneut aufgebaute Sitzung war also eine lebende Audioverbindung an einem eingefrorenen Wasserfall; und das Anhalten des Servers räumte ihn nicht mehr, weil jeder Hörer Sekunden später wieder da war und die Benutzerliste mit Sitzungen füllte, in denen in Wahrheit niemand saß. Die Seite verstummt und der Hörer lädt sie neu — genau das tat dieser Empfänger, bevor überhaupt ein Wiederverbinden eingebaut wurde.

Für die Limits dieses Handbuchs fällt damit auch eine Falle weg: gegen ein Ratenlimit würde ein automatischer Versuch genau die Sperre verlängern, die er zu umgehen sucht.

Die zwei Schließcodes

Code	Bedeutung	Wer ihn sendet
4003	abgewiesen — über einem Limit pro Adresse, oder Seite veraltet	<code>spectrumserver</code>
4001	vom Sysop hinausgeworfen	<code>spectrumserver</code> , wenn das Panel <code>/~~kick</code> aufruft

Beide liegen im privaten Bereich 4000–4999, können also nicht mit einem Protokollstatus verwechselt werden, und beide gelten auf jeder Seite als endgültig. Der Begründungstext reist im Schließ-Frame statt als Nachricht, weil das erste Frame auf `/audio` den Einstellungen des Empfängers vorbehalten ist und alles davor jede normale Verbindung zerstören würde.

Auf der Seite

Abgewiesen beim Laden der Seite — der Hörer kommt gar nicht erst in Gang, also erklären sich beide Seiten. Der Desktop zeigt ein Feld: „*Dieser Empfänger hat die Verbindung abgewiesen*“, den Grund und den Hinweis, dass alle, die sich einen Internetanschluss teilen, als eine Adresse zählen. `/mobile` zeigt dasselbe in seinem Hinweisbereich.

Abgewiesen oder hinausgeworfen mitten in der Sitzung — die Desktop-Seite hört einfach auf, bewusst ohne Meldung: die Zeichenschleife des Wasserfalls hält an, mehr nicht. `/mobile` zeigt eine Zeile, die zum Neuladen auffordert — außer nach einem Rauswurf, dann schweigt auch sie.

Dieses Schweigen nach einem Rauswurf ist Absicht. **Hinauswerfen heißt, dass der Sysop eine Sitzung beendet, es ist keine Strafe** — es wird nichts verkündet, und wer zurück will, lädt die Seite neu und ist eine Sekunde später ein ganz gewöhnlicher Hörer. Das Einzige, was der Code 4001 bringt, ist, dass der Browser weiß: das war keine abgebrochene Verbindung — der Rauswurf greift also, statt sich selbst aufzuheben.

Den Rauswurf erledigt `spectrumserver` selbst, nicht `proxy.py`: Hörer verbinden sich direkt mit dem Port des Empfängers, der Proxy besitzt also keine ihrer Verbindungen und sein Rauswurf erreicht niemanden. Das Panel ruft `/~kick?ip=&secs=` auf — nur über Loopback erreichbar und anhand des echten TCP-Gegenübers geprüft, nicht anhand von `X-Forwarded-For`, das vom Client kommt. `secs` ist standardmäßig null, also keine Sperre, nimmt aber eine Dauer an, wenn die Adresse doch eine Weile draußen bleiben soll; das nutzt dieselbe Sperrmechanik wie `per_ip_rate`, sodass der Wiederverbindungsversuch schon an der Tür abgewiesen wird.

8. Logs und Zähler lesen

Der Server

```
grep Refused spectrumserver.log
```

```
Refused /audio from 203.0.113.5: too many simultaneous connections
Refused /audio from 198.51.100.7: too many connection attempts
```

Steht dort nichts, wird niemand abgewiesen. Die Protokollierung ist auf höchstens eine Zeile je Adresse alle fünf Sekunden gedrosselt — ohne das würde ein Angriff zu unbegrenzten Schreibzugriffen werden, und das ist nur ein langsamerer Weg, die Station lahmzulegen.

Der Kernel

```
sudo ./setup-firewall.sh --status
```

Lesen Sie den `counter packets N` jeder Regel; das ist die Zahl der tatsächlich verworfenen Pakete. Lauter Nullen heißt, dass nichts blockiert wurde und Ihre Schwellen bequem locker sitzen.

Lassen Sie sich von den `elements = { ... }`-Listen in den Sets oberhalb der Regeln nicht beunruhigen. Das sind Ihre ganz normalen Hörer, die gegen die Limits *verfolgt* werden — ein Eintrag entsteht bei der ersten Verbindung einer Adresse und verfällt ein paar Minuten später. Verfolgen ist nicht Blockieren.

9. Ihre Zahlen wählen

Raten ist unnötig; Ihre eigenen Logs sagen es Ihnen. Das ist die Messung, die hier `per_ip=3` festgelegt hat, über eine Woche `logs/users_*.jsonl`:

Gleichzeitige Sitzungen von einer Adresse	Anzahl Adressen
1	682
2	54
3	13
mehr als 3	8

758 verschiedene Adressen in sieben Tagen, und ein Limit von drei hätte acht davon berührt. Die Flut saß bei 181.

Um das auf Ihrer eigenen Station zu wiederholen, ordnen Sie die Sitzungs-IDs je Adresse in `logs/users_*.jsonl` einander zu und nehmen die größte Überlappung. Beachten Sie: das Log führt `tune`- und `disconnect`-Ereignisse, aber kein `connect`-Ereignis, der Beginn einer Sitzung muss also aus ihrem ersten `tune` erschlossen werden.

Denken Sie an NAT. Ein Verein, eine Schule, ein Büro oder ein Mobilfunkanbieter erscheinen als eine einzige Adresse, und alle dahinter teilen sich ein Kontingent. Das steckt in jedem Limit, das auf eine Adresse geschlüsselt ist; die einzige Abmilderung ist, eine Zahl zu wählen, mit der Sie leben können. Meldet ein Hörer je, dass er nicht verbinden kann, ist das der erste Verdacht — prüfen Sie `grep Refused` vor allem anderen.

10. Fallen, die man kennen sollte

Niemals eine Frist für eine untätige Verbindung

Es gibt keinen `handshake_s`-Schalter mehr, und das ist der Grund. Eine Frist von 30 Sekunden für untätige Verbindungen sieht nach der naheliegenden Abwehr gegen einen Slowloris aus. Sie wurde gebaut, eingeschaltet und lieferte innerhalb einer Minute das hier:

```
Idle deadline: closed 1 connection(s) idle past 30s: 192.87.173.88
```

192.87.173.88 ist `etgd-websdr.ewi.utwente.nl` — **der Rückruf-Host von websdr.org selbst.** Er verbindet sich zu Ihrem Server zurück und bleibt dann untätig, deutlich länger als 30 Sekunden, bevor er sein `GET /~orgstatus` schickt. Die Frist schnitt ihn jedes Mal ab und die Registrierung hörte auf. Die Einstellung wurde entfernt statt als Option belassen: ein Selbstschuss, der Ihren Verzeichniseintrag zerstört, ist die sechzig Zeilen nicht wert, die ihn umsetzen.

Dieselbe Überlegung gilt für `open_handshake_timeout` von `websocketpp`, das es weiterhin gibt.

Der Zehn-Minuten-Wert in `spectrumserver.cpp` sieht absurd aus und ist tragend: er ist es, der dem Rückruf von websdr.org das Untätigsein überhaupt erlaubt. Es gibt noch einen zweiten Grund — der `/~orgstatus`-Handler antwortet aus einem Thread, der ein `dup()` des Sockets hält, und `websocketpp` beendet eine abgelaufene Verbindung mit `shutdown()`, einem Aufruf auf Socket-Ebene, der durch das Duplikat hindurchgreift und den Rückruf auch dann zerstören würde, wenn die Untätigkeit kein Thema wäre.

Wenn Sie stille Verbindungen begrenzen müssen, begrenzen Sie ihre Anzahl, nicht ihr Alter. Genau das sind `idle_per_ip` und `idle_total`.

Eine `nftables`-Datei muss idempotent sein

`nft -f` **hängt an** eine bereits vorhandene Tabelle an, statt sie zu ersetzen. Eine Regeldatei ohne `delete table`-Präambel verdoppelt daher die gesamte Kette bei jedem Neuladen — und genau das tut eine naive `systemd`-Unit bei jedem Dienstneustart. `setup-firewall.sh` erzeugt die Präambel für Sie; schreiben Sie eigene Regeln, machen Sie es genauso.

```
sudo ./setup-firewall.sh --status | grep -c dport # sollte 8 ergeben, nicht 16
```

11. Alles wieder abschalten

Die Server-Limits: Schlüssel auf 0 setzen oder aus der `.toml` löschen und den Empfänger neu starten. Es gibt keinen weiteren Zustand; nichts überdauert einen Neustart.

Die Firewall:

```
sudo ./setup-firewall.sh --remove
```

Das löscht die Tabelle, entfernt das gespeicherte Regelwerk und deaktiviert den Boot-Dienst. Ihre Maschine kehrt genau in den Zustand zurück, in dem sie vorher war — was auf den meisten Installationen bedeutet: gar keine Firewall. Das ist es wert, vorher bedacht zu werden.

KAPITEL 6

Mehrere Empfänger

MULTI_RECEIVER.md

Mehrere Empfänger — Sysop-Handbuch

Zwei oder mehr Empfänger auf einem Rechner, mit einer Empfängerauswahl auf der Seite.

Seit v5.0.0 kann ein PhantomSDR-Plus-Rechner mehrere Empfänger gleichzeitig betreiben — zum Beispiel einen RX-888 für Kurzwelle und einen RTL-SDR für 2 m — und die Hörer wechseln mit Schaltflächen im Seitenkopf zwischen ihnen, so wie OpenWebRX seine Profile anbietet. Jeder Empfänger behält seinen eigenen Wasserfall, Chat, seine Markierungen, Hörerliste und Stationsangaben; niemand auf einem Empfänger wird je von jemandem auf einem anderen gestört.

Eilig? Eine Station mit einem Empfänger braucht nichts davon und merkt keinen Unterschied. Für einen zweiten Empfänger: `instances/<name>/` mit eigener `config.toml` anlegen, mit `INSTANCE=<name> ./start-<radio>.sh` starten, beide Empfänger in `receivers.toml` eintragen und den Proxy neu starten. Das Beispiel in [Abschnitt 3](#) geht es Zeile für Zeile durch.

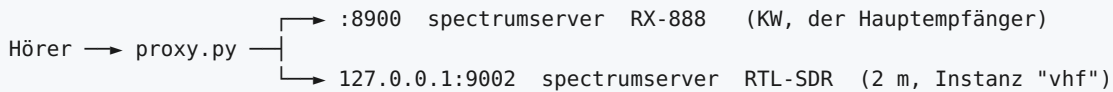
Am einfachsten: Führen Sie `./add-receiver.sh` aus (die Installer bieten es am Ende an). Das Skript fragt, welcher Empfänger und was er abdecken soll, installiert seinen Treiber, schreibt alles, was in [Abschnitt 3](#) und [Abschnitt 4](#) beschrieben ist, und bietet an, den Empfänger zu starten und den Proxy neu zu starten. Der Rest dieses Handbuchs erklärt, was es tut — für den Fall, dass Sie etwas von Hand ändern möchten.

Inhalt

1. Wie es funktioniert
2. Zwei Arten, die Empfänger zu veröffentlichen
3. Einen zweiten Empfänger hinzufügen
4. `receivers.toml`
5. Starten und Stoppen
6. Was die Hörer sehen
7. Das S-Meter oberhalb von 30 MHz
8. Sicherheit
9. Wie viele Empfänger passen
10. Fallen, die man kennen sollte

1. Wie es funktioniert

Jeder Empfänger ist ein vollständiger eigener spectrumserver mit eigenem SDR-Frontend, eigener Konfiguration und eigenem internen Port. Sie teilen niemals Abtastwerte, Einstellungen oder Hörer. Verbunden werden sie durch `proxy.py`, den Reverse-Proxy, der bereits das Admin-Panel ausliefert: Er liest `receivers.toml` und schickt jede Anfrage an einen Empfänger.



Der Proxy entscheidet anhand dieser Merkmale, in dieser Reihenfolge:

1. `?rx=<id>` in der Adresse — `http://ihr.host:8899/?rx=vhf`. Die Seite hängt es an jeden Socket, jede Abfrage und jeden Link an, den sie öffnet, damit ein Tab bei seinem Empfänger bleibt.
2. Das Cookie `rx`, das der Proxy setzt, sobald eine Anfrage ein gültiges `?rx=` trägt.
3. Der Host-Header, wenn ein Empfänger diesen Namen unter `hostnames` führt.
4. Sonst der Standardempfänger.

Der **Hauptempfänger** ist der wie gewohnt gestartete (`./start-rx888mk2.sh`). Jeder weitere Empfänger ist eine **benannte Instanz**: mit `INSTANCE=<name>` gestartet, in `instances/<name>/` konfiguriert und mit eigenem Log (`logwebsdr-<name>.txt`), Server-Log, Watchdog-Sperre und FIFO. Die Prozesse werden an einer Kennung `PHANTOMSDR_INSTANCE` in ihrer Umgebung unterschieden, nicht an ihrem Namen; zwei Empfänger dürfen also dasselbe Programm ausführen — zweimal `rtl_sdr`, oder `rx_sdr` für einen RSP1A und einen Airspy — und ein Neustart des einen berührt den anderen nie.

2. Zwei Arten, die Empfänger zu veröffentlichen

A — der Hauptempfänger behält seinen eigenen Port. Der Hauptempfänger bleibt genau, wo er war (etwa `:8900`), seine Hörer merken nichts, und die anderen Empfänger erreicht man über den Port des Proxys (`proxy_port` in `admin_config.json`, etwa `:8899`) mit `?rx=`. Nichts muss umziehen, nichts geht vom Netz; dafür gibt es zwei öffentliche Ports.

```

KW: http://ihr.host:8900/
2 m: http://ihr.host:8899/?rx=vhf

```

B — ein öffentlicher Port für alles. Der Proxy übernimmt den öffentlichen Port, und der Hauptempfänger zieht auf einen internen um. Setzen Sie `[front] port` in `receivers.toml`, verlegen Sie `[server] port` des Hauptempfängers und setzen Sie dort `host = "127.0.0.1"`, stellen Sie `public_port` in `admin_config.json` auf den neuen internen Port und tragen Sie `public_port = <der öffentliche Port>` unter `[websdr]` ein, damit das Verzeichnis weiterhin den richtigen meldet. Danach liegt alles hinter einem Port; der Preis ist eine kurze Unterbrechung beim Umzug.

```

KW: http://ihr.host:8900/
2 m: http://ihr.host:8900/?rx=vhf

```

Die Empfängerauswahl funktioniert in beiden Fällen gleich. Variante A ist der sicherere Anfang; B kann später folgen, ohne am zweiten Empfänger etwas zu ändern.

3. Einen zweiten Empfänger hinzufügen

add-receiver.sh erledigt alle folgenden Schritte — starten Sie es und beantworten Sie seine Fragen; es installiert auch den Treiber (`setup-rtlsdr.sh` für einen RTL-SDR, das passende `setup-*.sh` für die anderen), wählt einen freien internen Port und benennt den Empfänger nach dem, was er abdeckt. Die Schritte stehen hier ausgeschrieben, damit Sie seine Arbeit prüfen oder später ändern können.

Das Beispiel fügt einen RTL-SDR Blog V4 für 2 m als Instanz `vhf` hinzu, neben einem RX-888, der Port 8900 behält (Variante A).

1. Der Treiber. Der RTL-SDR Blog V4 braucht den Treiber von RTL-SDR Blog statt des Pakets `rtl-sdr` der Distribution, und der DVB-T-Treiber des Kernels muss den Stick freigeben. `install.sh` (Empfängeroption 2, „RTL-SDR Blog V4: ja“) erledigt beides. Prüfen Sie mit `rtl_test -t`; dort muss RTL-SDR Blog V4 Detected stehen.

2. Der Instanzordner. Alles, was zum Empfänger gehört, liegt in `instances/vhf/`; dieser Ordner wird nie committet und von keinem Update überschrieben:

```
instances/vhf/
├─ config.toml           seine spectrumserver-Konfiguration (Pflicht)
├─ instance.env          optional: Empfängerparameter und CPU-Bindung
├─ markers.json          seine eigenen Markierungen
├─ www/                  eigene Kopien der stationsbezogenen Dateien der Seite
│   └─ site_information.json
│   └─ wf-message.json
```

spectrumserver läuft mit `instances/vhf/` als Arbeitsverzeichnis; Chatverlauf, Markierungen, FFTW-Wisdom und `logs/` gehören damit ebenfalls dem Empfänger.

3. config.toml. Gehen Sie von `config-rtl.toml` aus und ändern Sie:

```
[server]
port=9002                # eigener interner Port
host="127.0.0.1"         # nur über den Proxy erreichbar
html_root="www/"         # seine wenigen eigenen Dateien...
html_fallback_root="../../frontend/dist/" # ...und für den Rest die gemeinsame Seite

[input]
sps=2400000              # muss zu -s in instance.env passen
frequency=145000000      # muss zu -f in instance.env passen

[input.defaults]
frequency=145500000
modulation="FM"

[kiwi_emulation]
enabled = false          # siehe die Fallen unten
```

Mit `html_fallback_root` behält die Instanz nur ihre eigene `site_information.json` und `wf-message.json`, während alles andere — die gebaute Seite selbst — aus `frontend/dist` kommt; ein einziger Frontend-Build erreicht so jeden Empfänger. Eine per Symlink verlinkte Kopie von `frontend/dist` funktioniert nicht: Der Server verweigert Dateien, die außerhalb seines eigenen Ordners liegen.

4. instance.env. Wird vom Startskript nach seinen eigenen Einstellungen gelesen:

```
RX_ARGS="-f 145000000 -s 2400000 -g 29.7 -" # -f und -s passen zu config.toml
SPECTRUM_CORES=8-11                        # weg von den Kernen des Hauptempfängers
```

5. Seine Seitenangaben. Kopieren Sie `frontend/site_information.json` nach `instances/vhf/www/` und passen Sie an: `siteReceiver`, `siteAntenna`, `siteSDRBaseFrequency` und `siteSDRBandwidth` (diese bestimmen, welche Bandschaltflächen erscheinen) sowie die in [Abschnitt 6](#) beschriebenen Schlüssel.

6. Starten.

```
INSTANCE=vhf ./start-rtl.sh
```

7. In receivers.toml eintragen und den Proxy neu starten (`sudo systemctl restart phantomsdr-proxy`). Der zweite Empfänger ist jetzt unter `http://ihr.host:8899/?rx=vhf` erreichbar, und beide Seiten zeigen die Auswahl.

4. receivers.toml

Kopieren Sie `receivers.toml.example` nach `receivers.toml`. Ohne diese Datei bedient der Proxy genau wie bisher einen einzigen Empfänger.

```
# [front]
# port = 8900          # nur Variante B: der Proxy hört zusätzlich auf diesem öffentlichen Port

[[receiver]]
id      = "hf"
name    = "HF 0-30 MHz (RX-888 MkII)"
port    = 8900
default = true
launcher = "start-rx888mk2.sh"
url      = "http://ihr.host:8900/"

[[receiver]]
id      = "vhf"
name    = "2 m (RTL-SDR Blog V4)"
port    = 9002
launcher = "start-rtl.sh"
instance = "vhf"
url      = "http://ihr.host:8899/?rx=vhf"
```

Schlüssel	Bedeutung
<code>id</code>	Kurzname für <code>?rx=</code> ; Buchstaben, Ziffern, <code>-</code> und <code>_</code>
<code>name</code>	Die Beschriftung der Schaltfläche des Empfängers
<code>port</code>	<code>[server] port</code> seines spectrumservers
<code>host</code>	Optional; wo dieser spectrumserver läuft (Standard <code>127.0.0.1</code>)
<code>default</code>	Der Empfänger für Anfragen, die keinen nennen; höchstens einer
<code>launcher</code>	Sein <code>start-*.sh</code> , für <code>start-all.sh</code> ; weglassen bei einem Empfänger, den dieser Rechner nicht startet
<code>instance</code>	Sein Instanzname; beim Hauptempfänger weglassen
<code>url</code>	Wohin seine Schaltfläche den Hörer schickt; ohne Angabe <code>/?rx=<id></code> auf dem Proxy
<code>hostnames</code>	Optionale DNS-Namen, die direkt zu diesem Empfänger führen

Der Proxy liefert die Liste auch als `/receivers.json` aus (IDs, Namen und Links — nie die internen Ports); diese Datei liest die Auswahl.

5. Starten und Stoppen

Befehl	Wirkung
<code>./start-rx88mk2.sh</code>	Startet oder startet den Hauptempfänger neu
<code>INSTANCE=vhf ./start-rtl.sh</code>	Startet oder startet den Empfänger <code>vhf</code> neu
<code>./start-all.sh</code>	Startet jeden Empfänger in <code>receivers.toml</code> , der einen <code>launcher</code> hat
<code>./stop-websdr.sh vhf</code>	Stoppt nur <code>vhf</code>
<code>./stop-websdr.sh main</code>	Stoppt nur den Hauptempfänger
<code>./stop-websdr.sh</code>	Stoppt jeden Empfänger, wie schon immer

Solange ein Empfänger gestoppt ist, verschwindet seine Schaltfläche aus der Auswahl — der Proxy führt nur die Empfänger auf, die antworten, und offene Seiten lesen die Liste einmal pro Minute neu — und sie kehrt zurück, sobald der Empfänger wieder läuft; läuft nur noch ein Empfänger, verschwindet die Zeile *Receivers*: ganz. Ein gestoppter Empfänger kommt allerdings von selbst zurück, sobald alle Empfänger gestartet werden: `./start-all.sh`, **Restart** im Admin-Panel und ein Neustart durch den Temperaturschutz starten alles, was in `receivers.toml` steht. Soll einer dauerhaft aus bleiben, entfernen Sie seinen Block `[[receiver]]` (oder kommentieren ihn aus) und starten den Proxy neu.

Stellen Sie im Admin-Panel das **Default start script** auf `start-all.sh`. Sein Stop hält bereits jeden Empfänger an, und Restart sowie der Temperaturschutz verwenden das Startskript; mit `start-all.sh` kehren deshalb alle Empfänger zurück statt nur des Hauptempfängers.

Der RADE-Helfer gehört nur zum Hauptempfänger. Eine benannte Instanz startet oder stoppt ihn nie.

6. Was die Hörer sehen

Die Empfängerauswahl. Auf einer Station mit mehr als einem Empfänger bekommt der Seitenkopf eine eigene Zeile, *Receivers*:, mit einer Schaltfläche pro Empfänger; der aktuelle erscheint gelb. Die Seite `/mobile` hat dieselben Schaltflächen als zweite Reihe in ihrer oberen Leiste, und sie öffnen die `/mobile`-Seite des anderen Empfängers.

Die eigenen Angaben jedes Empfängers. Eine Seite, die auf einem zweiten Empfänger geöffnet wird, lädt vor dem Start dessen `site_information.json`; Bandschaltflächen, Startfrequenz, Hörerliste, Stationsangaben sowie die Links bei Receiver und Antenna gehören damit zu ihm. Ein normaler Besuch beim Hauptempfänger stellt keine zusätzliche Anfrage.

Die Schlüssel in `site_information.json` dafür:

Schlüssel	Bedeutung
<code>siteReceiverId</code>	Der Empfänger, zu dem diese Seite gehört (seine <code>id</code>); beim Hauptempfänger weglassen
<code>siteReceiversList</code>	Wo die Auswahl die Liste liest, z. B. <code>http://ihr.host:8899/receivers.json</code> ; nötig bei einem Empfänger, dessen Seite nicht über den Proxy ausgeliefert wird
<code>siteReceiverURL</code>	Wohin der Name bei <i>Receiver</i> in <i>Open Additional Info</i> verlinkt
<code>siteAntennaURL</code>	Wohin der Name bei <i>Antenna</i> verlinkt; <code>" "</code> zeigt den Namen ohne Link

Wann die Auswahl verborgen bleibt. Die Auswahl zeigt nur eine Liste, die den Host nennt, unter dem die Seite geöffnet wurde. Über die LAN-Adresse geöffnet (`http://192.168.1.10:8900/`) bleibt sie verborgen; als `http://ihr.host:8900/` erscheint sie. Das ist Absicht: Eine Station, die die

`site_information.json` einer anderen unverändert übernommen hat, zeigt deren Empfänger nie als ihre eigenen.

7. Das S-Meter oberhalb von 30 MHz

Nach dem Standard der IARU Region 1 liegt S9 unterhalb von 30 MHz bei -73 dBm und **oberhalb bei -93 dBm**, in beiden Fällen sechs dB je S-Stufe. Die S-Meter (Zeigerinstrument, digitaler Balken und der Balken auf /mobile) folgen der eingestellten Frequenz: Oberhalb von 30 MHz zeigen sie VHF-S-Stufen, darunter wie bisher. Die Werte in dBm und dBµV werden nie verändert. Sollen die Instrumente auf einem leeren Kanal auf 0 ruhen wie bei einem VHF/UHF-Funkgerät, tragen Sie `"siteSMeterGateDb": 6` in die `site_information.json` des Empfängers ein (kein Neubau nötig): Oberhalb von **60 MHz** schlagen Zeiger und Balken dann erst aus, wenn ein Signal so viele dB über dem Rauschen in seiner eigenen Umgebung liegt (der stärkste Punkt im Durchlassbereich gegen das Spektrum ± 100 kHz darum herum). Das ist **standardmäßig aus** — ein kalibriertes Instrument, das das echte Bandrauschen zeigt, ist die ehrlichere Anzeige. Die Seite /mobile, die keinen Wasserfall hat, schätzt das Rauschen aus dem empfangenen Pegel selbst.

Kalibrieren Sie die dBm-Anzeige eines VHF-Empfängers mit `analog_smeter_offset` (Zeiger und Zahlen) und `smeter_offset` (digitaler Balken) unter `[input]` in seiner `config.toml` und starten Sie diesen Empfänger danach neu. Stellen Sie zuerst die Verstärkung ein: Ein RTL-SDR meldet Pegel relativ zu seinem eigenen Vollausschlag, jede Änderung der Verstärkung verschiebt also die Anzeige. Ohne Messsender sollte ein 50-Ω-Abschluss statt der Antenne, in USB mit 2,7-kHz-Filter, etwa -136 dBm anzeigen (thermisches Rauschen in 2,7 kHz ist $-139,7$ dBm, plus die Rauschzahl des Sticks).

Für einen RTL-SDR schreibt `add-receiver.sh` eine Startkalibrierung: eine feste Verstärkung `-g 29.7` in `instance.env` (die automatische Verstärkung des Tuners lässt sich nicht kalibrieren) sowie `analog_smeter_offset=-55` und `smeter_offset=-38`, gemessen an einem RTL-SDR Blog V4 bei dieser Verstärkung mit einem Signal von -71 dBm (63 µV). Ein anderer Stick desselben Modells liegt meist nur wenige dB daneben; prüfen Sie ihn mit einem bekannten Signal und kalibrieren Sie neu, wenn Sie die Verstärkung ändern.

8. Sicherheit

Den Proxy vor die Hörer zu stellen, hat zwei Lücken geschlossen; beide sind in v5.0.0 behoben:

- **Der Sysop-Kick über den Proxy.** `spectrumserver` erlaubt `/~~kick` nur vom eigenen Rechner, und für ihn kommt jede vom Proxy weitergeleitete Anfrage vom eigenen Rechner. Wer den Proxy-Port erreichen konnte, konnte deshalb jeden Hörer trennen und sperren. Der Proxy beantwortet `/~~kick` jetzt nur noch für Clients auf demselben Rechner.
- **Gefälschte Client-Adressen.** `spectrumserver` glaubte einen `X-Forwarded-For`-Header von jedem; ein Besucher konnte sich als `127.0.0.1` ausgeben und an den Limits pro Adresse vorbeigehen. Der Header wird jetzt nur noch von einem lokalen Proxy geglaubt, und der Proxy verwirft jede Kopie, die ein Client mitschickt, bevor er seine eigene anfügt.

Binden Sie jeden Empfänger, der über den Proxy erreicht wird, an `127.0.0.1 ([server] host)`, damit sein Port nicht am Proxy vorbei genutzt werden kann. Die Limits pro Adresse aus [Verbindungslimits](#) gelten je Empfänger.

9. Wie viele Empfänger passen

Die Software setzt keine Grenze; die Hardware tut es. Schmalbandige Empfänger sind genügsam: Ein RTL-SDR mit 2,4 Msps braucht etwa 5–10 % eines Kerns, einige Dutzend MB Speicher und etwa 38 Mbit/s auf dem USB.

- **USB 2.0 ist die übliche Grenze.** Alle USB-2-Geräte teilen sich einen Bus mit 480 Mbit/s, auch in einer blauen USB-3-Buchse. Drei bis vier RTL-SDR je Bus sind ein sicherer Wert; fügen Sie sie einzeln hinzu und beobachten Sie jeden Wasserfall auf Lücken.
- **Strom:** Ab mehr als zwei Sticks einen aktiven Hub verwenden.
- **Ein zweiter Breitbandempfänger** (ein weiterer RX-888 oder ein HackRF mit 20 Msps) ist etwas anderes: Er konkurriert mit dem ersten um USB 3, CPU und GPU.
- **Hörer kosten mehr als Empfänger.** Jeder Hörer kostet CPU auf seinem Empfänger und Upload-Bandbreite; die Gesamtzahl der Hörer zählt deshalb mehr als die Zahl der Empfänger.

10. Fallen, die man kennen sollte

- **Zwei Sticks desselben Modells haben dieselbe Seriennummer.** Jeder RTL-SDR Blog V4 meldet 00000001. Geben Sie jedem eine eigene mit `rtl_eeprom -s <seriennummer>` (immer nur ein Stick eingesteckt) und nennen Sie sie in `instance.env` (`RX_ARGS="-d <seriennummer> ..."`), sonst können die Sticks nach einem Neustart die Empfänger tauschen.
- **Kiwi-Clients können keinen Empfänger wählen.** Ein KiwiSDR-Client wählt eine nackte Adresse und einen Port; über den Proxy landet er immer beim Standardempfänger. Lassen Sie `[kiwi_emulation]` bei den anderen ausgeschaltet.
- **Cookies gelten pro Host, nicht pro Port.** Ein Besucher, der den zweiten Empfänger auf `:8899` benutzt hat, trägt `rx=vhf` auch zu `:8900`. Deshalb entscheidet eine Seite anhand von `siteReceiverId`, welcher Empfänger sie ist, nie anhand des Cookies.
- **Verzeichniseinträge.** Geben Sie jedem Empfänger in `[websdr]` einen eigenen Namen und setzen Sie bei einem Empfänger hinter dem Proxy `[websdr] public_port` auf den Port, den die Hörer benutzen, sonst wirbt der Eintrag mit dem internen.
- **Ein Frontend-Build zeigt kurz „Not Found“.** `frontend/dist` wird an Ort und Stelle neu gebaut; einige Sekunden lang fehlt die Seite während eines Builds bei jedem Empfänger.
- **Nichts startet die Empfänger beim Hochfahren,** außer Sie richten es ein. Nach einem Neustart bringt `./start-all.sh` alle zurück.

KAPITEL 7

Statistikserver

sdr-stats/README.md

Installationsanleitung für System Stats

Eine umfassende Anleitung, um Ihrer PhantomSDR-Anwendung mithilfe des automatisierten Installationsskripts eine Serverüberwachung in Echtzeit hinzuzufügen.

Übersicht

Diese Funktion fügt Ihrer PhantomSDR-Oberfläche eine **Stats**-Schaltfläche hinzu, die Systeminformationen in Echtzeit anzeigt:

- CPU-Auslastung, Kerne, Frequenz, Temperatur und Top-Prozesse
- Speichernutzung (belegt/gesamt/Prozent)

Die Statistiken werden automatisch alle 5 Sekunden aktualisiert, solange das Modalfenster geöffnet ist.

Nicht zu verwechseln mit der Seite „Graphen“ des Admin-Panels. Dieses Fenster zeigt eine Momentaufnahme, die sich beim Öffnen aktualisiert und keinen Verlauf speichert. Das Admin-Panel hat eine eigene Seite **Graphen**, die CPU-Takt, Last, Temperatur und Nutzer online alle 2 Sekunden erfasst und über die letzten 15 Minuten bis 24 Stunden darstellt — siehe das [Admin-Panel-Handbuch](#).

Voraussetzungen

- **Linux-Server** (Ubuntu/Debian empfohlen)
- **Internetverbindung** (zum Herunterladen von Node.js und Paketen)
- **Sudo-Zugriff** (für die Einrichtung des systemd-Dienstes)
- **PhantomSDR-Plus** bereits installiert und in Betrieb
- **Im-sensors** (empfohlen für Intel/AMD-CPU-Temperatur – wird vom Skript automatisch installiert)

Installationsmethoden

HINWEIS

Das Hauptinstallationsskript bietet dies bereits an. `./install.sh` — und die vier Distributionsskripte — führen `install-stats-server.sh` standardmäßig im Rahmen der normalen PhantomSDR-Plus-Installation aus, auf einer frischen Maschine ist der Statistikserver also meist schon eingerichtet. Nutzen Sie diese Seite, um ihn separat zu installieren oder um Port, Adresse oder Dienst später zu ändern.

Option 1: Automatisierte Installation (Empfohlen)

Das automatisierte Skript erledigt alles für Sie.

Schritt 1: Das Skript ausführen

```
cd ~/PhantomSDR-Plus
chmod +x install-stats-server.sh # only if it is not already executable
./install-stats-server.sh
```

WICHTIG: Führen Sie es NICHT als root oder mit sudo aus. Führen Sie es als Ihr regulärer Benutzer aus.

Schritt 2: Den interaktiven Aufforderungen folgen

Das Skript führt Sie durch:

1. Node.js-Installation (falls erforderlich)

```
△ Node.js / npm is not installed
Would you like to install Node.js 22 via nvm now? (y/n): y
```

Das Skript installiert Node.js 22 über nvm, auf jeder Linux-Distribution — ohne root und ohne apt-Quelle. Dieselbe Frage erscheint, wenn Node.js zwar vorhanden, aber älter als 22 ist.

2. Installationsverzeichnis

```
Installation directory [/home/user/sdr-stats-server]:
```

Drücken Sie die Eingabetaste für die Standardeinstellung oder geben Sie einen benutzerdefinierten Pfad an.

3. Port-Konfiguration (Verbessert!)

```
Port Configuration:
  ▫ Default port: 3001
  ▫ Common alternatives: 8080, 5000, 8888
Enter port number [3001]:
```

Das Skript validiert Ihren Port:

- • Prüft, ob es sich um eine gültige Zahl handelt
- • Warnt, wenn Port < 1024 (erfordert Root-Rechte)
- • Validiert den Bereich (1-65535)
- • Prüft, ob der Port bereits verwendet wird
- • Zeigt an, welcher Prozess ihn belegt
- • Ermöglicht die Wahl eines anderen Ports

Beispielszenarien:

Standardport verwenden:

```
Enter port number [3001]: ↵
✓ Port 3001 selected
```

Benutzerdefinierten Port wählen:

```
Enter port number [3001]: 8080
✓ Port 8080 selected
```

Port bereits belegt:

```

Enter port number [3001]: 3001
△ Port 3001 is currently in use
Process using port 3001:
node 12345 user 20u IPv6 123456 TCP *:3001 (LISTEN)
Choose a different port? (y/n): y
Enter port number [3001]: 3002
✓ Port 3002 selected

```

Ungültiger Port:

```

Enter port number [3001]: abc
x Port must be a number
Enter port number [3001]: 99999
x Port must be between 1 and 65535
Enter port number [3001]: 3001
✓ Port 3001 selected

```

4. Serveradresse

```

What is your server's public address?
Examples: Your_site_IP, 192.168.1.100, localhost
Server address: Your_site_IP

```

Geben Sie die öffentliche Domain oder IP-Adresse Ihres Servers ein.

5. Bestätigung

```

Please confirm your settings:
  Installation directory: /home/user/sdr-stats-server
  Port: 3001
  Server address: Your_site_IP
  Stats URL will be: http://Your_site_IP:3001
Continue with these settings? (y/n): y

```

6. Einrichtung des Systemd-Dienstes (Optional)

```

Would you like to set up the server as a system service (auto-start on boot)? (y/n): y

```

Empfohlen: Wählen Sie **y**, damit der Server beim Booten automatisch startet.

Schritt 3: Installation abgeschlossen! ✓

Das Skript zeigt eine Zusammenfassung an:

```

Installation Complete! ✓
Open the port in the router!

```

```

Configuration Summary:
  Installation: /home/user/sdr-stats-server
  Port: 3001
  Server: Your_site_IP
  API URL: http://Your_site_IP:3001/api/system-stats

```

Konfigurieren Sie Ihre PhantomSDR-Anwendung

Nachdem das Skript abgeschlossen ist, müssen Sie Ihre Svelte-Anwendung aktualisieren.

Schritt 1: `site_information.json` aktualisieren

Da `site_information.json` bereits während der ersten Einrichtung bearbeitet wurde, müssen Sie nur die neue `siteStats`-Zeile mit Ihrem Port hinzufügen:

```
{
  "siteSysop": "your name or callsign",
  "siteSysopEmailAddress": "mail@mail.net",
  "siteGridSquare": "QTH locator",
  "siteCity": "City Country",
  "siteInformation": "https://github.com/svlbtl/PhantomSDR-Plus",
  "siteHardware": "Hardware you are using, ",
  "siteSoftware": "Software you are using",
  "siteReceiver": "Receiver model",
  "siteAntenna": "Receiving Antenna.",
  "siteNote": "This is a bright new open-source WebSDR project, under active development.",
  "siteIP": "http://Your_site_IP:port",
  "siteStats": "http://Your_site_IP:3001", ← DIESE ZEILE HINZUFÜGEN (Ihren Port verwenden)
  "siteSDRBaseFrequency": 0,
  "siteSDRBandwidth": 30000000,
  "siteRegion": 1,
  "siteChatEnabled": true
}
```

Hinweis: Verwenden Sie den Port, den Sie während der Installation ausgewählt haben!

Schritt 2: `App.svelte` aktualisieren (die neueste GitHub-Version enthält die aktualisierte Datei bereits)

Nehmen Sie diese **4 Änderungen** an Ihrer `App.svelte`-Datei vor:

Änderung 1: Sicherstellen, dass `siteStats` deklariert ist (etwa Zeile 50)

```
import {
  siteSysop,
  siteSysopEmailAddress,
  siteInformation,
  siteGridSquare,
  siteCity,
  siteHardware,
  siteSoftware,
  siteReceiver,
  siteAntenna,
  siteNote,
  siteIP,
  siteStats, // ← DIESE ZEILE HINZUFÜGEN
  siteSDRBaseFrequency,
  siteSDRBandwidth,
  siteRegion,
  siteChatEnabled,
} from "../site_information.json";
```

Änderung 2: systemStats-Objekt aktualisieren (etwa Zeile 530)

```
let systemStats = {
  cpu: { usage: 0, cores: 0, temperature: null, frequency: null, topProcesses: [] }, // ←
  topProcesses: [] HINZUFÜGEN
  memory: { used: 0, total: 0, percent: 0 }
};
```

Änderung 3: Fetch-URL aktualisieren (etwa Zeile 541)

```
async function fetchSystemStats() {
  try {
    const response = await fetch(`${siteStats}/api/system-stats`); // ← DIESE ZEILE ÄNDERN
    if (response.ok) {
      systemStats = await response.json();
    } else {
      console.error('Failed to fetch system stats:', response.statusText);
    }
  } catch (error) {
    console.error('Error fetching system stats:', error);
  }
}
```

Änderung 4: Anzeige der Top-Prozesse hinzufügen (im CPU-Abschnitt, etwa Zeile 3838)

Fügen Sie diesen Code nach dem Temperaturabschnitt hinzu:

```
{#if systemStats.cpu.frequency}
<div style="display: flex; justify-content: space-between; margin-bottom: 0.25rem;">
  <span>Frequency:</span>
  <span style="color: #4ade80;">{systemStats.cpu.frequency.current} GHz{#if
systemStats.cpu.frequency.max} <span style="opacity: 0.7;">(max {systemStats.cpu.frequency.max})</
span>{/if}</span>
</div>
{/if}

{#if systemStats.cpu.temperature !== null}
<div style="display: flex; justify-content: space-between; margin-bottom: 0.25rem;">
  <span>Temperature:</span>
  <span style="color: {systemStats.cpu.temperature > 70 ? '#fbbf24' :
'#4ade80'};">{systemStats.cpu.temperature}°C</span>
</div>
{/if}

<!-- DIESEN GESAMTEN ABSCHNITT HINZUFÜGEN: -->
{#if systemStats.cpu.topProcesses && systemStats.cpu.topProcesses.length > 0}
<div style="margin-top: 0.75rem; padding-top: 0.75rem; border-top: 1px solid
rgba(255,255,255,0.1);">
  <h4 style="margin: 0 0 0.5rem 0; font-size: 0.85rem; color: rgba(0, 225, 255, 0.8);">Top
Processes:</h4>
  {#each systemStats.cpu.topProcesses as process}
    <div style="display: flex; justify-content: space-between; margin-bottom: 0.25rem; font-size:
0.85rem;">
      <span style="overflow: hidden; text-overflow: ellipsis; white-space: nowrap; max-width:
70%;">{process.name}</span>
      <span style="color: {process.cpu > 50 ? '#ef4444' : process.cpu > 25 ? '#fbbf24' :
'#4ade80'};">{process.cpu}%</span>
    </div>
  {/each}
</div>
{/if}
```

Schritt 3: Anwendung neu kompilieren

```
cd /path/to/your/phantomsdr-app  
npm ./recompile.sh
```

Die Installation testen

Test 1: API-Endpunkt prüfen

```
curl http://localhost:3001/api/system-stats
```

Erwartete Ausgabe (JSON):

```
{  
  "cpu": {  
    "usage": 45.2,  
    "cores": 8,  
    "coresUsed": 1.3,  
    "temperature": 62.5,  
    "frequency": {  
      "current": 3.0,  
      "max": 3.9,  
      "limit": 4.4  
    },  
    "topProcesses": [  
      {"name": "node", "cpu": 12.5},  
      {"name": "phantomsdr", "cpu": 8.3}  
    ]  
  },  
  "memory": {  
    "used": 8.5,  
    "total": 16,  
    "percent": 53  
  }  
}
```

Test 2: Dienststatus prüfen

Wenn Sie es als Dienst installiert haben:

```
sudo systemctl status sdr-stats.service
```

Sollte anzeigen: Active: active (running)

Test 3: Im Browser testen

1. Öffnen Sie Ihre PhantomSDR-Weboberfläche
2. Scrollen Sie zum Abschnitt **Additional Info**
3. Klicken Sie auf die Schaltfläche **Open Additional Info**
4. Suchen Sie die Schaltfläche **Stats** neben den PC-Infos
5. Klicken Sie darauf, um das Statistik-Modalfenster zu öffnen
6. Überprüfen Sie, ob die Statistiken angezeigt und aktualisiert werden

Dienstverwaltung

Wenn Sie es als systemd-Dienst installiert haben, verwenden Sie diese Befehle:

```
# Start the service
sudo systemctl start sdr-stats.service

# Stop the service
sudo systemctl stop sdr-stats.service

# Restart the service (after updating files)
sudo systemctl restart sdr-stats.service

# Check status
sudo systemctl status sdr-stats.service

# View real-time logs
sudo journalctl -u sdr-stats.service -f

# View last 50 log entries
sudo journalctl -u sdr-stats.service -n 50

# Enable auto-start on boot
sudo systemctl enable sdr-stats.service

# Disable auto-start
sudo systemctl disable sdr-stats.service

# Edit service
sudo nano /etc/systemd/system/sdr-stats.service

# Delete service
sudo rm /etc/systemd/system/sdr-stats.service
```

Manuelle Installation (Option 2)

WICHTIG

Für eine normale Installation brauchen Sie diesen Abschnitt nicht. `install-stats-server.sh` erledigt all das für Sie und wird vom Hauptinstallationskript standardmäßig ausgeführt. Folgen Sie ihm nur für eine manuelle Einrichtung oder um einen Teil von Hand zu reparieren.

⚠️ Warnung: Bei der manuellen Installation werden möglicherweise veraltete Dateien verwendet. Das **automatisierte Skript wird dringend empfohlen**, da es:

- Die neueste Version mit verbesserter Temperaturerkennung installiert
- Im-sensors automatisch installiert und konfiguriert
- Alle Abhängigkeiten automatisch verwaltet

Wenn Sie dennoch die manuelle Installation bevorzugen:

1. Node.js installieren

```
# Any distribution - nvm needs no root and no apt source.
# NOT NodeSource: deb.nodesource.com now answers HTTP 403 on every path.
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.4/install.sh | bash
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
nvm install 22 && nvm use 22 && nvm alias default 22
```

2. lm-sensors installieren (Erforderlich für Intel/AMD-Temperatur)

```
sudo apt-get update
sudo apt-get install -y lm-sensors
sudo sensors-detect --auto
```

3. Verzeichnis erstellen

```
mkdir ~/sdr-stats-server
cd ~/sdr-stats-server
```

4. Neueste Dateien herunterladen

Wichtig: Die Server-Dateien liegen nicht mehr als Kopien im Repository - das Installationsskript erzeugt sie.

Führen Sie stattdessen das automatisierte Skript einmal aus, um die neuesten Dateien zu generieren, und kopieren Sie sie dann:

```
# Run the automated script from the repository root
cd ~/PhantomSDR-Plus
./install-stats-server.sh
# When prompted, use a temporary port like 9999
# After installation completes, copy the generated files
cp ~/sdr-stats-server/system-stats-server.js ~/your-manual-install-dir/
cp ~/sdr-stats-server/package.json ~/your-manual-install-dir/
```

ODER erstellen Sie die Dateien manuell mit dem neuesten Code aus dem Installationsskript.

5. Port bearbeiten (falls erforderlich)

```
nano ~/sdr-stats-server/system-stats-server.js
```

Ändern Sie Zeile 6:

```
const PORT = process.env.PORT || 3001; // Change this number
```

6. Abhängigkeiten installieren

```
npm install
```

7. Testlauf

```
npm start
```

8. Dienst einrichten (Optional)

Folgen Sie der systemd-Dienst-Einrichtung aus dem Abschnitt zur automatisierten Installation.

Den Port nach der Installation ändern

Wenn Sie den Port nach der Installation ändern müssen:

Schritt 1: Die Serverdatei bearbeiten

```
nano ~/sdr-stats-server/system-stats-server.js
```

Ändern Sie Zeile 6:

```
const PORT = process.env.PORT || 3001; // Change this number
```

Schritt 2: site_information.json aktualisieren

```
"siteStats": "http://Your_site_IP:NEW_PORT"
```

Schritt 3: Den Dienst neu starten

```
sudo systemctl restart sdr-stats.service
```

Schritt 4: Ihre Svelte-App neu erstellen

```
cd /path/to/phantomsdr-app  
npm run build
```

Fehlerbehebung

Problem: Port bereits belegt

Fehler:

```
Error: listen EADDRINUSE: address already in use :::3001
```

Lösung:

1. Finden Sie heraus, was den Port verwendet:

```
sudo lsof -i :3001
```

1. Stoppen Sie diesen Prozess:

```
kill <PID>  
# or  
sudo systemctl stop sdr-stats.service
```

1. Starten Sie den Dienst erneut:

```
sudo systemctl start sdr-stats.service
```

Alternative: Führen Sie das Installationsskript erneut aus und wählen Sie bei der Aufforderung einen anderen Port.

Problem: Der Dienst startet nicht

Fehler:

```
status=217/USER
```

Lösung:

1. Überprüfen Sie die Dienstdatei:

```
cat /etc/systemd/system/sdr-stats.service
```

1. Überprüfen Sie, ob die Zeile `User=` mit Ihrem Benutzernamen übereinstimmt:

```
whoami
```

1. Bearbeiten Sie sie bei Bedarf:

```
sudo nano /etc/systemd/system/sdr-stats.service
```

1. Neu laden und neu starten:

```
sudo systemctl daemon-reload
sudo systemctl restart sdr-stats.service
```

Problem: Statistiken werden im Browser nicht angezeigt

Symptome: Das Modalfenster öffnet sich, aber es gibt keine Daten oder es werden 0-Werte angezeigt

Lösungen:

1. Überprüfen Sie die Browser-Konsole (F12) auf Fehler
2. Überprüfen Sie, ob die API erreichbar ist:

```
curl http://localhost:3001/api/system-stats
```

1. Überprüfen Sie, ob der Dienst läuft:

```
sudo systemctl status sdr-stats.service
```

1. Überprüfen Sie, ob `siteStats` in `site_information.json` mit Ihrer Serveradresse und Ihrem Port übereinstimmt
2. Überprüfen Sie auf CORS-Fehler – stellen Sie sicher, dass der Server Ihre Domain zulässt

Problem: Temperatur zeigt null an

Ursache: Temperaturmessung auf Ihrem System nicht verfügbar

Lösungen:

1. **Für die meisten Linux-Systeme:** Überprüfen Sie, ob die thermische Zone existiert:

```
cat /sys/class/thermal/thermal_zone0/temp
```

1. lm-sensors installieren:

```
sudo apt-get install lm-sensors
sudo sensors-detect
sensors
```

1. **Hinweis:** Die Temperatur ist möglicherweise nicht verfügbar auf:
2. Virtuellen Maschinen
3. Einigen VPS-Anbietern
4. Windows Subsystem for Linux (WSL)
5. Nicht-Linux-Systemen

Das ist normal – der Rest der Statistiken funktioniert trotzdem!

Problem: Statistikwerte stimmen nicht mit `top` überein

Ursache: Der CPU-Abtastzeitraum war in älteren Versionen zu kurz

Lösung: Die aktuelle Version verwendet eine 1-Sekunden-Abtastung für genaue Messwerte. Wenn Sie eine alte Installation haben:

1. Aktualisieren Sie die Serverdatei:

```
nano ~/sdr-stats-server/system-stats-server.js
```

1. Suchen Sie Zeile 18 und ändern Sie:

```
}, 100); // Old value
```

in:

```
}, 1000); // New value (1 second)
```

1. Neu starten:

```
sudo systemctl restart sdr-stats.service
```

Oder führen Sie einfach das Installationsskript erneut aus, um die neueste Version zu erhalten.

Problem: Node.js nicht gefunden

Fehler:

```
node: command not found
```

Lösung:

1. Node.js installieren:

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.4/install.sh | bash
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
nvm install 22 && nvm use 22 && nvm alias default 22
```

1. Überprüfen:

```
node --version
npm --version
```

Was Sie sehen werden

Nach Installation und Konfiguration zeigt das Statistik-Modalfenster Folgendes an:

CPU-Abschnitt

- **Auslastung:** Gesamt-CPU-Auslastung in Prozent
- **Kerne:** Wie viel CPU tatsächlich genutzt wird, ausgedrückt in ganzen Kernen — z. B. `1.3 / 12 in use` heißt, die Maschine leistet die Arbeit von 1,3 Kernen. Der Wert ist exakt und braucht keine Schwelle. Es ist bewusst keine Zählung belegter Kerne: der Scheduler verteilt die Arbeit eines Kernels auf viele, sodass eine solche Zählung diese Zahl immer nur annähern könnte.
- **Frequenz:** Durchschnittlicher aktueller CPU-Takt über alle Kerne in GHz, mit dem schnellsten Kern in Klammern (falls verfügbar)
- **Temperatur:** CPU-Temperatur in Celsius (falls verfügbar), farblich codiert: grün bis 69 °C, orange 70-79 °C, rot ab 80 °C
- **Top-Prozesse:** Die 5 CPU-intensivsten Prozesse mit farbcodierten Prozentwerten:
 - Grün: bis 50 % CPU
 - Orange: 51-80 % CPU
 - Rot: über 80 % CPU

Speicherabschnitt

- **Belegt:** Belegter Speicher (GB)
- **Gesamt:** Gesamter Systemspeicher (GB)
- **Auslastung:** Speicherauslastung in Prozent (farbcodiert)

🔄 Automatische Aktualisierung

- Die Statistiken werden alle **5 Sekunden** aktualisiert, solange das Modalfenster geöffnet ist
- Die Aktualisierung stoppt, wenn Sie das Modalfenster schließen (spart Ressourcen)

Temperaturerkennung

Der Statistikserver verwendet eine fortschrittliche Temperaturerkennung mit mehreren Methoden:

Unterstützte Systeme:

- **Intel-Prozessoren:** Liest von coretemp-Sensoren über lm-sensors
- **AMD-Prozessoren:** Liest von k10temp-Sensoren (Tdie/Tctl)
- **ARM-Prozessoren:** Liest von thermischen Zonen (Raspberry Pi usw.)

Anforderungen:

- **Intel/AMD:** Erfordert das Paket `lm-sensors` (wird vom Skript automatisch installiert)
- **ARM/Raspberry Pi:** Funktioniert sofort, keine zusätzlichen Pakete erforderlich

Installationspriorität:

1. Versucht den Befehl `sensors` für die Package-/Core-Temperatur (Intel/AMD)
2. Greift auf das direkte Lesen von `coretemp` zurück (Intel)
3. Greift auf `thermal_zone0` zurück (ARM/Raspberry Pi)

Wenn die Temperatur `null` anzeigt, installieren Sie `lm-sensors`:

```
sudo apt-get install lm-sensors
sudo sensors-detect --auto
sudo systemctl restart sdr-stats.service
```

Funktionsübersicht

Echtzeitüberwachung - Live-Systemstatistiken von Ihrem Server **CPU-Metriken** - Auslastung, Kerne, Temperatur, Top-Prozesse **Speicherverfolgung** - Belegt, gesamt und Prozent
Automatische Aktualisierungen - Aktualisiert alle 5 Sekunden **Farbcodiert** - Visuelle Indikatoren für Warnungen **Ressourcenschonend** - Minimaler Ressourcenverbrauch **Einfache Installation** - Das automatisierte Skript erledigt alles **Port-Validierung** - Intelligente Portauswahl mit Konflikterkennung **Dienstverwaltung** - Automatischer Start beim Booten

Sicherheitshinweise

- Der Statistikserver akzeptiert Verbindungen von jeder Herkunft (CORS: *)
- Erwägen Sie für die Produktion, CORS auf Ihre Domain zu beschränken
- Die Portauswahl validiert die Eingabe und prüft auf Konflikte
- Das Lesen der CPU-Temperatur erfolgt schreibgeschützt und sicher
- Es werden keine sensiblen Systeminformationen offengelegt

Support & Ressourcen

Enthaltene Dateien:

In diesem Verzeichnis (das Installationsskript selbst liegt im Wurzelverzeichnis als `PhantomSDR-Plus/install-stats-server.sh` - es gibt immer nur eine Kopie):

- `package.json` - Abhängigkeiten, identisch mit dem, was das Skript schreibt
- Dieses Dokument, auch auf `de`, `el`, `es`, `fr`, `hr` und `ru` verfügbar

Vom Installationsskript in `~/sdr-stats-server/` erzeugt:

- `system-stats-server.js` - Backend-Server, mit dem gewählten Port erzeugt
- `package.json` - Abhängigkeiten
- `node_modules/` - von `npm install` installiert

Hilfe erhalten:

- Sehen Sie sich den obigen Abschnitt zur Fehlerbehebung an
- Überprüfen Sie die Dienstprotokolle: `sudo journalctl -u sdr-stats.service -n 50`
- Testen Sie die API manuell: `curl http://localhost:3001/api/system-stats`
- Überprüfen Sie die Konfigurationsdateien

Statistikserver deinstallieren:

- Löschen Sie den Ordner `/sdr-stats-server`
- Wenn Sie einen Dienst erstellt haben, löschen Sie ihn einfach mit:

```
sudo rm /etc/systemd/system/sdr-stats.service`
```

- Bearbeiten Sie `App.svelte`: Finden Sie den Codeabschnitt:

```

        <!-- In case you don't want the Stats Button to appear, please comment this
button section (12 lines)-->
        <!-- System Stats Button -->
        <button
            type="button"
            class="glass-button text-white py-1 px-2 ml-2 rounded text-xs"
            on:click={openSystemStats}
            title="System Resources"
            aria-haspopup="dialog"
            aria-expanded={showSystemStats}
            aria-controls="system-stats-dialog"
            style="color:rgba(0, 225, 255, 0.993); font-size: 0.75rem;"
        >
            Stats
        </button>
    ...
    und ersetzen Sie ihn durch:
```

```

        <!-- In case you don't want the Stats Button to appear, please comment this button
section (12 lines)-->
        <!-- System Stats Button -->
        <!--
        <button
            type="button"
            class="glass-button text-white py-1 px-2 ml-2 rounded text-xs"
            on:click={openSystemStats}
            title="System Resources"
            aria-haspopup="dialog"
            aria-expanded={showSystemStats}
            aria-controls="system-stats-dialog"
            style="color:rgba(0, 225, 255, 0.993); font-size: 0.75rem;"
        >
            Stats
        </button>
        -->
```

```
...
```

- Kompilieren Sie mit dem Befehl neu:

```

    cd PhantomSDR-Plus
    ./recompile.sh
    \\\

- Starten Sie den PC neu und starten Sie den Server

---

**Autor:** Erstellt für PhantomSDR-Plus
**Version:** 1.1 (Verbesserte CPU-Temperaturerkennung für Intel/AMD) **Aktualisiert:** Januar 2026
**Lizenz:** MIT

---

## Kurzreferenz der Befehle

\\\bash
# Installation
cd ~/PhantomSDR-Plus
chmod +x install-stats-server.sh
./install-stats-server.sh

# Service Management
sudo systemctl start sdr-stats.service      # Start
sudo systemctl stop sdr-stats.service       # Stop
sudo systemctl restart sdr-stats.service    # Restart
sudo systemctl status sdr-stats.service     # Status
sudo journalctl -u sdr-stats.service -f     # Logs

# Testing
curl http://localhost:3001/api/system-stats # Test API
curl http://localhost:3001/api/health      # Health check

# Troubleshooting
sudo lsof -i :3001                          # Check port
whoami                                       # Check username
node --version                             # Check Node.js

```

Viel Erfolg beim Überwachen!

KAPITEL 8

Decoder

DECODERS.md

PhantomSDR-Plus — Benutzerhandbuch der Decoder

Dieses Handbuch behandelt alle in PhantomSDR-Plus integrierten Decoder. Alle Decoder werden auf dieselbe, nachfolgend beschriebene Weise aktiviert; danach folgen die Einrichtungsanweisungen für jeden einzelnen Decoder.

Inhaltsverzeichnis

1. Einen Decoder starten
2. FT8
3. FT4-FT2
4. JS8
5. CW — Morsetelegrafie
6. QRSS Grabber
7. WSPR
8. HF-FAX / WEFAX
9. NAVTEX
10. FSK / RTTY — einschließlich PSK31 und Olivia
11. SSTV
12. Allgemeine Tipps

1. Einen Decoder starten

Alle Decoder erreichen Sie über den Abschnitt **Decoder Options** unterhalb der Bedienelemente des Audiospektrogramms im Hauptfenster.

Schritte:

1. Klicken Sie auf die Schaltfläche **Decoder: OFF**, um sie auf **Decoder: ON** umzuschalten (sie wird blau, wenn sie aktiv ist).
2. Öffnen Sie das Auswahlmenü, das rechts neben der Schaltfläche erscheint, und wählen Sie den gewünschten Decoder.
3. Das Bedienfeld des gewählten Decoders erscheint unterhalb der Bedienelemente — folgen Sie den decoderspezifischen Anweisungen im entsprechenden Abschnitt dieses Handbuchs.
4. Um die Dekodierung zu beenden, wählen Sie — **Select decoder** — aus dem Menü, oder klicken Sie auf **Decoder: ON**, um wieder auf OFF zu schalten.

Decoder-Tasten mit einem Druck

Das Dropdown-Menü ist nicht der einzige Weg. Das Hauptpanel enthält eine Tastenreihe **Decoders** — direkt unter **Wheel Tuning Steps** — mit einer Taste je Decoder:

Taste	Decoder	Taste	Decoder
FT8	FT8	SSTV	SSTV
FT4	FT4	NAVTEX	NAVTEX
FT2	FT2	RTTY	FSK / RTTY
CW	CW		
WSPR	WSPR	FAX	HF FAX / WEFAX

Ein Tastendruck erledigt den ganzen Ablauf: Er wählt den Decoder aus, schaltet den Decoder ON und holt sein Fenster in den sichtbaren Bereich. Die Taste leuchtet blau, solange ihr Decoder läuft. **Ein erneuter Druck auf dieselbe Taste schaltet den Decoder ab** — sein Fenster schließt sich mit.

RADEL und **RADEU** fehlen in dieser Reihe absichtlich. Sie sind Digital-Voice-Betriebsarten und keine Text-Decoder, deshalb haben sie ein eigenes Tastenpaar neben der Überschrift **Modes selector** sowie in den Popup-Fenstern **Modes** und **Bands**. Diese verhalten sich genau gleich — drücken zum Starten, erneut drücken zum Stoppen. Siehe das [RADE-Handbuch](#).

Die Tasten, das Dropdown-Menü und die ON/OFF-Taste steuern denselben Zustand — was Sie auch benutzen, das Übrige folgt.

Es kann immer nur ein Decoder aktiv sein. Der Wechsel zu einem anderen Decoder beendet den vorherigen automatisch.

Der **QRSS-Grabber** gehört nicht zu diesem Auswahlmenü — er hat einen eigenen Bereich **QRSS** und kann gleichzeitig mit einem Decoder laufen.

Die Dekodierung läuft im Hintergrund. SSTV, HF-FAX, NAVTEX, FSK/RTTY und CW laufen jeweils in einem eigenen Web-Worker-Thread, sodass die Dekodierarbeit nie mit der Audiowiedergabe oder dem Wasserfall konkurriert. Das Starten oder Beenden eines Decoders unterbricht den Ton nicht, und die Oberfläche bleibt reaktionsschnell, während ein Bild oder eine Seite empfangen wird. Alle Decoder erhalten Rohaudio, das *vor* AGC, Rauschminderung und Stummschaltung abgegriffen wird — den Empfänger stummschalten oder diese Einstellungen nach Gehör zu verändern, wirkt sich also nicht auf die Dekodierung aus.

Ein laufender Decoder besitzt Betriebsart und Durchlassbereich. Normalerweise folgt die Betriebsart dem Bandplan in `bands-config.js`: Führt die Abstimmung in einen als LSB oder AM gekennzeichneten Abschnitt, schaltet der Empfänger um. Solange ein Decoder läuft, geschieht das nicht mehr. Der Decoder hält die Betriebsart, die er braucht (USB bei den meisten, eine eigene bei RADE), und den Durchlassbereich, den er braucht — PSK31 etwa ± 100 Hz, Olivia seine volle Bandbreite, RTTY seinen Shift — und beides überlebt ein Umstimmen, auch den Sprung auf ein anderes Band. Ohne das würde FT8 auf 40 m den Empfänger auf LSB umschalten, sobald die Abstimmung bewegt wird, und die schmalen Decoder-Durchlassbereiche würden wieder auf das volle SSB-Filter aufgehen.

Die bandeigene Betriebsart kehrt in dem Moment zurück, in dem Sie den Decoder abschalten. Von Hand können Sie die Betriebsart jederzeit überstimmen: Die Mode-Schaltflächen sind eine bewusste Entscheidung und setzen sich immer durch. Der CW-Decoder ist die Ausnahme von alldem — er dekodiert in der Betriebsart, in der Sie gerade hören, und übernimmt den Empfänger nie.

Decoder ID — die Betriebsart automatisch erkennen

Über den Decoder-Tasten liegt das Feld **Decoder ID**. Es beantwortet die Frage „Was höre ich da eigentlich?“, wenn Sie auf ein digitales Signal stoßen, das Sie nicht einordnen können, und bietet anschließend den passenden Decoder mit einem Klick an.

Es ist **standardmäßig ausgeschaltet** und kostet im ausgeschalteten Zustand nichts. Drücken Sie **On** und lassen Sie das Signal eingestellt. Etwa eine halbe Minute lang zeigt es *Listening...* mit einem Fortschrittsbalken — die Zeitmessungen müssen zwei vollständige FT8-Zyklen sehen, bevor sie aussagekräftig sind — und benennt dann die Betriebsart:

Decoder ID [On] NAVTEX / SITOR-B 99% 170 Hz shift · 100.00 Bd · 43 dB S/N [Use NAVTEX]

- Die **Betriebsart** und ein **Vertrauenswert**. Grün über 75 %, gelb über 50 %, orange darunter — eine orange Antwort ist ein Hinweis, kein Urteil.
- **Was tatsächlich gemessen wurde**: belegte Bandbreite, Tonabstand, Symbolrate, Burst-Länge, das passende UTC-Raster, Signal-Rausch-Abstand. Das ist die Beweisgrundlage hinter dem Urteil, damit Sie selbst urteilen können.
- **Use** startet den richtigen Decoder, genau so, als hätten Sie dessen Taste gedrückt — und richtet dabei den Empfänger auf diese Betriebsart ein. Bei den Betriebsarten im FSK-Panel wird zusätzlich die richtige Variante gewählt — RTTY, Wetter-RTTY, PSK31 oder Olivia.
- Die nächstplatzierten Kandidaten stehen nach **or**. Ein Klick probiert stattdessen diesen aus — eine falsche erste Antwort kostet Sie also nichts.
- ******** bedeutet, dass die eingestellte Frequenz eine bekannte Anruffrequenz für diese Betriebsart ist und dass dies in die Bewertung eingeflossen ist.

Es schaltet Ihren Decoder niemals selbst um. Es schlägt immer nur vor. Und sobald Sie einen Decoder starten — über die Tastenreihe, über die Auswahlliste oder über die Schaltfläche **Use** der Decoder ID — schaltet sich die Decoder ID selbst ab: ihre Aufgabe war es, einen auszuwählen, und der Decoder braucht nun dasselbe Audiosignal.

Es richtet den Empfänger auf die Betriebsart ein. Eine Betriebsart zu wählen heißt, sie auch arbeiten zu wollen, also folgt der Empfänger: Die Abstimmfrequenz rückt in die Mitte des Wasserfalls und die Ansicht wird rund 100 kHz breit, das Seitenband wird das, auf dem diese Betriebsart gearbeitet wird, und der Durchlassbereich verengt sich auf das Fenster, in dem diese Betriebsart lebt — das gesamte 3-kHz-Teilband für FT8, FT4, FT2 und JS8, 1350–1650 Hz für WSPR, 250–750 Hz für NAVTEX, 800–2700 Hz für HF-FAX, 900–2600 Hz für SSTV und ± 250 Hz beiderseits der Abstimmfrequenz für CW, wobei zusätzlich auf CW umgeschaltet wird. Ein schmaler Durchlassbereich hält die Nachbarsignale aus dem Decoder heraus — genau das, was Sie von Hand getan hätten. Zwei Ausnahmen: SSTV behält das Seitenband, das es selbst gewählt hat — auf 80 und 40 m üblicherweise LSB, wo ein erzwungenes USB die Tonzuordnung umkehren würde — und die FSK-Betriebsarten behalten das Fenster, das ihr eigenes Panel aus Shift oder Bandbreite der Variante ableitet und das enger ist als jeder feste Wert. Ein Druck auf **Use** beim bereits laufenden Decoder schaltet ihn weiterhin ab; der Empfänger bleibt dann genau so, wie er war.

Die langsamen Betriebsarten brauchen länger. WSPR liegt auf einem Zwei-Minuten-Raster und SSTV-Bilder laufen ein bis zwei Minuten, deshalb werden diese beiden später benannt als die übrigen — geben Sie ihnen drei bis vier Minuten auf der Frequenz. HF-FAX und SSTV werden außerdem nur erkannt, wenn sie wie üblich abgestimmt sind — das Bildband liegt dann bei etwa 1500 bis 2300 Hz im NF-Bereich, dort, wo es auch ihre Decoder erwarten. Ein Frequenzwechsel startet die Messung von vorn, denn alles bisher Gesammelte beschreibt die gerade verlassene Frequenz.

Was es erkennt: FT8, FT4, JS8, WSPR, CW, NAVTEX/SITOR-B, RTTY mit 45,45 Bd, Wetter-RTTY (DWD), PSK31, Olivia, HF-FAX, SSTV und DSC. In dieser Version gibt es keinen DSC-Decoder, daher wird DSC benannt und mit *no decoder* gekennzeichnet.

Recentre & retry. Wo das Signal im NF-Durchlassbereich liegt, spielt zwischen etwa 400 Hz und 2700 Hz keine Rolle. Außerhalb dieses Bereichs verschlechtert sich die Messung, und eine gelbe Schaltfläche **Recentre & retry** erscheint. Ein Druck darauf verschiebt die Abstimmung so, dass das Signal in der

Mitte des Durchlassbereichs liegt, und startet eine neue Messung. Das ist ein einzelner berechneter Schritt, keine Suche.

Drei ehrliche Grenzen:

- **FT8, JS8 und FT2 sind dasselbe Signal.** Dieselbe 8-FSK-Modulation, derselbe Tonabstand, und JS8 verwendet bei Normal-Geschwindigkeit den 15-Sekunden-Takt von FT8. Am Signal selbst lassen sie sich nicht unterscheiden. An der Frequenz schon: auf einer FT8-Anruffrequenz meldet es FT8, auf einer JS8-Frequenz JS8, und überall sonst meldet es ehrlich die Familie — *FT8 / JS8 / FT2*. Starten Sie einen der beiden Decoder und sehen Sie, welcher Text liefert.
- **Es schweigt lieber, als zu raten.** Unterhalb von etwa 10 dB Signal-Rausch-Abstand meldet es *Nothing above the noise* oder *Not recognised*, statt eine Betriebsart zu benennen, die es nicht wirklich erkennen kann. Das ist Absicht: eine selbstbewusst falsche Antwort ist schlimmer als gar keine.
- **Ein schwankendes Signal wird anders gelesen.** WSPR sendet 110 Sekunden ohne Unterbrechung, also länger als ein gewöhnlicher QSB-Einbruch dauert. Ein tiefer Schwund zerschneidet die Aussendung deshalb in Stücke, und ihr Zwei-Minuten-Takt lässt sich daraus nicht mehr ablesen. Er wird dann auf einem anderen Weg zurückgewonnen: Der gesamte Beobachtungszeitraum wird gegen das Zwei-Minuten-Raster geprüft statt gegen einzelne Aussendungen. In der Messzeile steht dann *120 s grid (through fading)*. Das ist ein schwächerer Beleg als eine vollständig gehörte Aussendung, deshalb ist die angezeigte Sicherheit bewusst geringer. Ein Schwund, der das Signal ganz im Rauschen versinken lässt, ist gar nicht mehr zu retten. Olivia ist strenger als die übrigen: Sie wird erst benannt, wenn ihre Symbolrate gemessen ist, und verstummt daher früher.

2. FT8

Worum es geht: FT8 ist eine weit verbreitete digitale Schwachsignal-Betriebsart, die Funkamateure weltweit nutzen. Aussendungen dauern genau 15 Sekunden, und das Protokoll kann Signale bis 20–25 dB unter dem Rauschteppich aufnehmen. Es ist die am häufigsten genutzte Betriebsart für Weitverbindungen (DX).

Empfohlene Frequenzen (USB)

Band	Frequenz
160 m	1.840 MHz
80 m	3.573 MHz
40 m	7.074 MHz
30 m	10.136 MHz
20 m	14.074 MHz
17 m	18.100 MHz
15 m	21.074 MHz
12 m	24.915 MHz
10 m	28.074 MHz

Einrichtung

1. Stimmen Sie auf eine der oben genannten FT8-Frequenzen ab und stellen Sie die Betriebsart auf **USB**.
2. Aktivieren Sie den Decoder und wählen Sie **FT8** aus dem Menü. Oder drücken Sie einfach die Taste **FT8**.
3. Das Bedienfeld **FT8 Messages** erscheint automatisch darunter.

Die Ausgabe lesen

Die Meldungsliste zeigt dekodierte Aussendungen, sobald sie eintreffen. Jeder 15-Sekunden-Zyklus liefert einen neuen Schwung Meldungen. Das Feld **Farthest** (oben rechts im Bedienfeld) zeigt die größte in der laufenden Sitzung dekodierte Entfernung in Kilometern.

Typisches Meldungsformat: **CQ DX AA1BB FN31** — ein CQ-Ruf des Rufzeichens AA1BB aus dem Planquadrat FN31.

Die **SNR** bei jeder Dekodierung ist ein echter Signal-Rausch-Abstand, gemessen am dekodierten Signal gegenüber dem umgebenden Rauschen und auf 2500 Hz Bandbreite bezogen, so wie WSJT-X sie angibt. Schwache Stationen zeigen negative Werte bis etwa -20 dB, starke positive; auf denselben Aufnahmen stimmen beide Programme im Mittel überein, bei einer einzelnen Dekodierung typisch auf etwa 3 dB. FT4 und FT2 verwenden dasselbe Verfahren.

Nach einem Update: Der FT8/FT4/FT2-Decoder ist eine eigene Datei, die Browser lange zwischenspeichern. Liegen alle SNR-Werte weiterhin zwischen etwa 0 und +5 dB, laden Sie die Seite hart neu (**Strg+Umschalt+R**, auf dem Mac **Cmd+Umschalt+R**).

Tipp: FT8 ist eng zeitsynchronisiert. Ihr Browser nutzt die Uhr Ihres Rechners; weicht die Systemzeit um mehr als ein paar Sekunden ab, schlägt die Dekodierung fehl. Halten Sie Ihre Systemzeit per NTP synchron.

3. FT4-FT2

Worum es geht: FT4 ist eine schnellere Variante von FT8, gedacht für Contest-Betrieb. Jeder Sendezyklus dauert 7,5 Sekunden (die Hälfte von FT8), wodurch die Betriebsart doppelt so schnell ist, aber ein etwas stärkeres Signal benötigt. FT2 ist eine noch schnellere Variante von FT8. Es handelt sich um eine ultraschnelle 77-Bit-Betriebsart mit TR-Perioden von 3,75 Sekunden (= die Hälfte von FT4) — noch experimentell.

Empfohlene Frequenzen (USB)

Band	Frequenz FT4
80 m	3.575 MHz
40 m	7.047 MHz
30 m	10.140 MHz
20 m	14.080 MHz
15 m	21.140 MHz
10 m	28.180 MHz

Band	Frequenz FT2
160 m	1.843 bis 1.846
80 m	3.578 bis 3.581
60 m	5.360 (regionale Vorschriften beachten)
40 m	7.052 bis 7.062
30 m	10.144
20 m	14.084
17 m	18.108
15 m	21.144
12 m	24.923
10 m	28.184

Einrichtung

1. Stimmen Sie auf eine FT4-Frequenz ab und stellen Sie die Betriebsart auf **USB**.
2. Aktivieren Sie den Decoder und wählen Sie **FT4** oder **FT2** aus dem Menü. Oder drücken Sie einfach die Taste **FT4 / FT2**.
3. Das Bedienfeld **FT4 Messages** bzw. **FT2 Messages** erscheint darunter, im Aufbau identisch mit dem FT8-Bedienfeld.

Hinweis: FT4 und FT8 verwenden unterschiedliche Spektralformate und sind nicht austauschbar. Achten Sie darauf, dass Sie auf einer FT4-Frequenz stehen, wenn Sie diesen Decoder nutzen.

4. JS8

Was es ist: JS8 (der Modus von JS8Call) nimmt die Schwachsignal-Engine von FT8 und macht daraus einen Tastatur-zu-Tastatur-Gesprächsmodus. Während FT8 feste Nachrichten mit 13 Zeichen sendet, überträgt JS8 freien Text in kleinen Portionen und fügt aufeinanderfolgende Aussendungen wieder zu ganzen Sätzen zusammen — alles über etwa zwölf Zeichen kommt also über mehrere 15-Sekunden-Zyklen an. Die Empfindlichkeit entspricht etwa der von FT8, sodass er dort funktioniert, wo Sprache und CW nicht mehr durchkommen.

Empfohlene Frequenzen (USB)

Band	Frequenz
160m	1.842 MHz
80m	3.578 MHz
40m	7.078 MHz
30m	10.130 MHz
20m	14.078 MHz
17m	18.104 MHz
15m	21.078 MHz
12m	24.922 MHz
10m	28.078 MHz

Geschwindigkeiten

JS8 kennt fünf Geschwindigkeiten. Alle Stationen eines Gesprächs müssen dieselbe verwenden. **Normal** ist die Anruf-Geschwindigkeit und trägt fast den gesamten Verkehr — dort beginnen.

Geschwindigkeit	Zyklus	Bandbreite	Wofür
Slow	30 s	25 Hz	Sehr schwache Wege; am empfindlichsten
Normal	15 s	50 Hz	Die übliche Anruf-Geschwindigkeit
Fast	10 s	80 Hz	Schnellerer Austausch, braucht stärkeres Signal
Turbo	6 s	160 Hz	Starke lokale Signale
Ultra	4 s	250 Hz	Experimentell, kaum auf dem Band zu sehen

Einrichtung

1. Auf eine JS8-Frequenz abstimmen und **USB** einstellen.
2. Decoder einschalten und **JS8** aus der Liste wählen, oder einfach die Taste **JS8** drücken.
3. Das Fenster **JS8 Decoder** erscheint darunter. **Speed** auf *Normal* stehen lassen, sofern die gesuchte Station nicht nachweislich eine andere benutzt.

Die Ausgabe lesen

Das Fenster hat zwei Listen.

Noch eintreffende Nachrichten stehen oben in Grün mit blinkendem Cursor. Eine JS8-Nachricht kann vier Zyklen dauern — bei Normal eine volle Minute — hier sieht man den Satz entstehen. Es ist kein Fehler, wenn das eine Weile so bleibt.

Fertige Nachrichten füllen die Hauptliste, je eine Zeile:

Spalte	Bedeutung
Mode	Immer JS8
Hz	NF-Frequenz des Signals im Durchlassbereich
dB	Signal-Rausch-Abstand in 2500 Hz Bezugsbandbreite
Message	Der dekodierte Text; Rufzeichen erscheinen grün

Typische Nachrichten:

- SV1BTL KM17: HB — ein Heartbeat: die Station meldet, dass sie auf dem Band ist, mit ihrem Locator.
- KN4CRD: K00G SNR -05 — eine gerichtete Nachricht: KN4CRD teilt K00G mit, dass er ihn mit -5 dB hört.
- MP 100W 8M/BALUN JN58KH AUGSBURG, MARTIN — freier Text, hier eine Stationsbeschreibung über vier Zyklen.

Eine **blass und kursiv** dargestellte Zeile ist abgelaufen, bevor ihr letzter Rahmen ankam. Der Text ist echt, kann aber abgeschnitten sein.

Sync offset

Der Regler **Sync offset** bestimmt, wie lange nach der UTC-Zyklusgrenze die Aufzeichnung beginnt, und gleicht damit die Laufzeit der Audiokette aus. **Auto** aktiviert lassen: der Decoder misst das Timing der empfangenen Signale und stellt sich selbst nach.

Spektrum-Markierungen

Der kleine Spektrumstreifen im Fenster zeichnet für jedes im letzten Zyklus dekodierte Signal eine gelbe senkrechte Linie, sodass sich die Hz-Spalte direkt am Display ablesen lässt.

Die eigenen Spots sehen

Wenn der Empfänger JS8-Spots an PSK Reporter meldet (siehe den Reiter Spot Reporting im Admin-Panel), öffnet die Schaltfläche **JS8 map** im Feld **Note**: die Karte von PSK Reporter, gefiltert auf das Rufzeichen dieses Empfängers und den Modus JS8 — genau wie **FT8 map** und **FT4 map** für jene Modi.

Tipp: JS8 ist viel ruhiger als FT8. An einem Nachmittag kann eine Aussendung alle paar Minuten normal sein, ebenso längere Pausen ganz ohne Betrieb. Geben Sie ihm fünf bis zehn Minuten, bevor Sie einen Fehler vermuten. 20m (14.078) und 40m (7.078) sind die üblichen Anlaufstellen.

Tipp: Wie FT8 ist JS8 streng zeitsynchron und nutzt die Uhr Ihres Rechners. Weicht die Systemzeit um mehr als ein bis zwei Sekunden ab, wird nichts dekodiert. Halten Sie sie per NTP synchron.

5. CW — Morsetelegrafie

Worum es geht: Der CW-Decoder horcht auf Morsesignale (Continuous Wave) und wandelt sie in Echtzeit in Text um. Er verfolgt die Signalfrequenz automatisch und passt sich der Gebegeschwindigkeit des Operators an.

Empfohlene Frequenzen

CW ist auf allen Amateurbändern aktiv, typischerweise im unteren Teil jedes Bandes. Übliche Stellen:

Band	Segment
40 m	7.000–7.040 MHz
20 m	14.000–14.070 MHz
15 m	21.000–21.080 MHz

Einrichtung

1. Stimmen Sie auf ein CW-Signal ab und verwenden Sie je nach Situation die Demodulationsart **CW** oder **CW-L**.

2. Aktivieren Sie den Decoder und wählen Sie **CW** aus dem Menü. Oder drücken Sie einfach die Taste **CW**.
3. Das Bedienfeld **CW Decoder** erscheint darunter.

Die Ausgabe lesen

- Die Kopfzeile des Bedienfelds zeigt die erkannte Signalfrequenz in Hz (z. B. ≈ 700 Hz) und die geschätzte Gebegeschwindigkeit in Wörtern pro Minute (z. B. $\cdot 22$ WPM).
- Wird kein Signal erkannt, zeigt die Kopfzeile **scanning...**
- Auf einem leeren Kanal wird nichts ausgegeben, und Geschwindigkeit und Ton werden nicht angezeigt. Der Decoder beginnt erst zu schreiben, wenn sich Taste gedrückt und Taste offen deutlich unterscheiden, was reines Rauschen nicht erzeugt, ein bei etwa -10 dB noch lesbares Signal aber reichlich.
- Der dekodierte Text läuft in bernsteinfarbener Festbreitenschrift durch. Der blinkende Cursor (█) markiert die Stelle, an der gerade geschrieben wird.
- Klicken Sie auf **Clear**, um den Ausgabepuffer zu löschen.

Tipps: - Zentrieren Sie Ihren Durchlassbereich auf den CW-Ton. Der Decoder arbeitet am besten, wenn das CW-Signal etwa zwischen 400 und 900 Hz im Audiospektrum liegt. - Geschwindigkeiten von etwa 8 bis 40 WPM werden gelesen, auch Farnsworth-Gebung (schnell gegebene Zeichen mit längeren Pausen dazwischen): Die Geschwindigkeitsverriegelung hält, und Buchstaben werden nicht in einzelne Wörter zerlegt. Stark handgetastetes (unregelmäßiges) Morsen kann die Genauigkeit weiterhin verringern. - Der Decoder liefert die besten Ergebnisse bei einem einzelnen sauberen Signal. Starkes QRM benachbarter Signale auf demselben Band kann ihn verwirren.

6. QRSS Grabber

Was es ist: QRSS ist CW, das so langsam gegeben wird, dass ein einzelner Punkt Sekunden statt Millisekunden dauert. Bei dieser Geschwindigkeit belegt das Signal nur einen Bruchteil eines Hertz, sodass eine entsprechend schmale Analyse die Spur noch 20–30 dB *unter* dem Rauschpegel sichtbar macht. Mit dem Ohr ist dabei nichts zu holen — QRSS wird **angeschaut**, nicht angehört. Der Grabber rechnet eine eigene, sehr lange FFT über das Empfängeraudio und zeichnet die klassische Grabber-Darstellung: Frequenz auf der senkrechten Achse, Zeit von links nach rechts laufend, die neueste Spalte immer am rechten Rand.

Das meiste, was Sie sehen werden, stammt von MEPT-Baken (Manned Experimental Propagation Transmitter) — Sender kleinster Leistung, oft ein paar hundert Milliwatt an einem Draht, die sich fortlaufend in sehr langsamem Morse oder in FSK-Mustern identifizieren.

Wo man suchen muss

QRSS-Baken leben in schmalen Fenstern von 100–200 Hz nahe dem unteren Bandende. Sie müssen sie nicht auswendig kennen: Der Grabber hat eine Liste **Band** mit den unten aufgeführten Fenstern, und eine Auswahl stimmt darauf ab und richtet den Empfänger dafür ein (siehe *Den Grabber öffnen*).

Band	Fenstermitte	Hinweis
30m	10.140,00 kHz	Das Haupt-QRSS-Fenster — mit Abstand das belebteste, Tag und Nacht
40m	7.039,90 kHz	
40m	7.000,85 kHz	Knights-Fenster, 7.000,8–7.000,9 kHz
20m	14.096,90 kHz	
80m	3.569,90 kHz	
80m	3.568,60 kHz	Älteres Fenster
80m	3.500,85 kHz	Knights-Fenster, 3.500,8–3.500,9 kHz
160m	1.837,90 kHz	
160m	1.843,30 kHz	Älteres Fenster
630m	476,10 kHz	
2200m	137,70 kHz	LF-QRSS / DFCW
60m	5.288,55 kHz	
17m	18.105,90 kHz	
15m	21.095,90 kHz	
12m	24.925,90 kHz	
10m	28.125,70 kHz	
10m	28.000,85 kHz	Älteres Fenster
10m	28.322,00 kHz	Alternative
6m	50.294,30 kHz	

Mehrere Bänder erscheinen doppelt, weil tatsächlich zwei Konventionen in Gebrauch sind. Die moderneren Fenster liegen 200 Hz unterhalb der WSPR-Frequenz des jeweiligen Bandes; die älteren *Knights*-Fenster liegen ganz woanders, auf 40 und 80 m deutlich tiefer. Ist ein Band auf dem einen still, versuchen Sie das andere.

Das sind Konventionen, keine Vorschriften, und einige sind regional — besonders die Werte für 10 m schwanken. Nehmen Sie die Liste als Ausgangspunkt und richten Sie sich nach der örtlichen Praxis.

Den Grabber öffnen

Der QRSS-Grabber steht **nicht** im Decoder-Auswahlmenü. Er hat einen eigenen Bereich **QRSS**, neben den Bedienelementen für Spektrogramm und Decoder.

1. **Show** anklicken. Die Grabber-Fläche erscheint, und die Bedienelemente klappen neben der Schaltfläche auf.
2. Ein Fenster aus der Liste **Band** wählen und **Tune** drücken. Das erledigt alles, was die Betriebsart braucht, in einem Schritt: Es stimmt auf das Fenster ab, schaltet den Empfänger auf **CW** und setzt einen Durchlassbereich, der gerade breit genug für den beobachteten Ausschnitt ist. Die Anzeige zeigt danach die echte QRSS-Frequenz, und die Spuren landen auf der Mittellinie der Darstellung.
3. **Speed** auf die Punktlänge der Bake einstellen.
4. **Centre** und **Span** wirken wie bisher; eine Änderung formt den Durchlassbereich mit, sodass Anzeige und Empfänger im Gleichschritt bleiben.
5. **Hide** anklicken, um zu beenden. Die normale Betriebsart des Bandes kehrt zurück.

Von Hand geht es weiterhin auch — in **USB** etwa 1 kHz unterhalb des Fensters abstimmen, damit die Spuren bei rund 800 Hz Audio landen. Die Liste **Band** erspart Ihnen nur die Rechnerei.

Solange ein Fenster aus der Liste abgestimmt ist, hält der Grabber den Empfänger so fest wie ein Decoder: CW und sein Durchlassbereich überleben ein Umstimmen, und das Fenster folgt der Abstimmung, sodass die Spur beim Absuchen des Bandes auf der Mittellinie bleibt. Er gibt den Empfänger frei, sobald Sie den Grabber ausblenden, eine Mode-Schaltfläche drücken oder einen Decoder starten.

Geschwindigkeiten

Einstellung	Punktlänge	Auflösung	Analysefenster	Neue Spalte alle
QRSS 3	3 s	0,73 Hz	1,4 s	0,7 s
QRSS 6	6 s	0,37 Hz	2,7 s	1,4 s
QRSS 10	10 s	0,18 Hz	5,5 s	2,7 s
QRSS 30	30 s	0,09 Hz	10,9 s	5,5 s
QRSS 60	60 s	0,05 Hz	21,8 s	10,9 s

Eine Einstellung, die schneller ist als die Bake, verschenkt Empfindlichkeit — die Spur wird dünn und verrauscht. Eine langsamere Einstellung verschmiert aufeinanderfolgende Punkte und Striche zu einem einzigen Balken. Das Panel startet mit **QRSS 6**, ein sinnvoller Ausgangspunkt bei einem unbekannten Signal: empfindlicher als QRSS 3, dabei schneller in der Anzeige und toleranter gegenüber Drift als QRSS 10. Sobald die Form der Tastung erkennbar ist, wechseln Sie auf die Geschwindigkeit, die die Bake tatsächlich sendet — jede Stufe schneller kostet 3 dB Empfindlichkeit.

Bedienelemente

Bedienelement	Wirkung
Band	Die oben aufgeführten QRSS-Fenster. Tune stimmt auf das gewählte ab, in CW mit passendem Durchlassbereich.
Speed	Legt die Transformationslänge fest — den Kompromiss zwischen Frequenz- und Zeitauflösung (Tabelle oben).
Centre	Audiofrequenz in der Mitte der Anzeige, 100–3000 Hz.
Span	Höhe des dargestellten Ausschnitts: 20, 50, 100 oder 200 Hz, beim Start 100 Hz — die Breite eines QRSS-Subbands. Ein schmalere Ausschnitt verteilt jede Spur auf mehr Pixel. Bei breiteren Ausschnitten fallen mehr Bins in das Panel, als es Pixelzeilen hat; jede Zeile zeigt dann das stärkste von ihr überdeckte Bin, sodass sich nichts zwischen den Zeilen verstecken kann.
Gain	–10 bis +40 dB. Verschiebt die Farbskala gegenüber dem gemessenen Rauschpegel; erhöhen, um schwache Spuren aufzuhellen.
Color	Rainbow, Green oder Grayscale.
Clear	Löscht die Fläche und setzt die Rauschreferenz zurück.

Die Anzeige lesen

Die Frequenzmarken laufen am linken Rand, die höchste Frequenz oben. Unter der Fläche zeigt eine Statuszeile die gewählte Geschwindigkeit, die tatsächlich verwendete Auflösung (z. B. 0.183 Hz/bin · 5.5 s window) und die Spaltenrate (z. B. 2.7 s/column).

Die Farbskala bezieht sich auf das **aktuelle** Rauschen, das laufend aus den leeren Teilen jeder Spalte gemessen wird, nicht auf einen absoluten Pegel. Änderungen an Lautstärke, AGC, Antenne oder Band

erfordern daher keine neue Gain-Einstellung — die Anzeige hält den Kontrast gegenüber dem jeweiligen Bandrauschen konstant. Wie bei den Decodern wird das Audio vor AGC, Rauschunterdrückung und Stummschaltung abgegriffen; Sie können den Empfänger also stummschalten und trotzdem weiter zusehen.

Was die Formen bedeuten:

- Ein **konstanter Träger** zeichnet eine gerade waagerechte Linie.
- **Langsames Morsen** zeichnet diese Linie als Kette kurzer und langer Abschnitte — Punkte und Striche, von links nach rechts zu lesen.
- **Gebogene oder wandernde Spuren** stammen von einem nicht stabilisierten Bakenoszillator beim Aufwärmen oder Abkühlen. Das ist normal, und oft ist gerade diese Driftsignatur das, woran ein regelmäßiger Grabber-Beobachter eine Station wiedererkennt.
- **Senkrechte Streifen** über den ganzen Ausschnitt sind Atmosphäris oder lokale Störimpulse, kein Signal.

Tipps: - Geduld. Bei QRSS 30 dauert eine einzelne Spalte 5,5 Sekunden, ein vollständiges Rufzeichen braucht also zehn Minuten oder mehr, um über den Schirm zu wandern. Einfach laufen lassen. - Beim Abstimmen über die Liste **Band** ist der Durchlassbereich um das Fenster bereits eingengt. Wer von Hand abstimmt, macht es selbst: Das ändert nichts an dem, was die Transformation auflöst, hält aber starke Nachbarn davon ab, die AGC des Empfängers zu bedienen. - Der Grabber ist von den Decodern unabhängig — Sie können ihn laufen lassen, während ein Decoder an etwas anderem arbeitet. - Ein breiter Span bei langsamer Geschwindigkeit ist die aufwendigste Kombination; die gesamte Transformation läuft im Browser, auf einer bescheidenen Maschine also besser 50 Hz Span wählen.

7. WSPR

Worum es geht: WSPR (Weak Signal Propagation Reporter, ausgesprochen „whisper“) ist eine Bakenbetriebsart für extrem schwache Signale, die KW-Ausbreitungswege weltweit kartiert. Jede Aussendung dauert etwa 110 Sekunden und passt in einen 200 Hz breiten Kanal. Der Decoder wartet auf einen vollständigen, an UTC ausgerichteten 2-Minuten-Zeitschlitz, bevor er dekodiert.

Empfohlene Frequenzen (USB, Anzeigefrequenz)

Band	Anzeigefrequenz
160 m	1.836.600 MHz
80 m	3.568.600 MHz
40 m	7.038.600 MHz
30 m	10.138.700 MHz
20 m	14.095.600 MHz
17 m	18.104.600 MHz
15 m	21.094.600 MHz

Einrichtung

1. Stimmen Sie auf eine der oben genannten WSPR-Anzeigefrequenzen ab und stellen Sie die Betriebsart auf **USB**.

2. Das WSPR-Signal belegt den Audiobereich 1400–1600 Hz. Die Auswahl von **WSPR** erledigt den Rest: Der Empfänger schaltet auf USB und der Durchlassbereich verengt sich auf 1350–1650 Hz, was den gesamten Suchbereich des Decoders abdeckt. Weitere Anpassungen sind nicht nötig.
3. Aktivieren Sie den Decoder und wählen Sie **WSPR** aus dem Menü. Oder drücken Sie einfach die Taste **WSPR**.
4. Das Bedienfeld **WSPR-2 Decoder** erscheint darunter.

Die Ausgabe lesen

Das Bedienfeld zeigt einen Fortschrittsbalken für den aktuellen 2-Minuten-Zeitschlitz:

- **Cyanfarbener Balken füllt sich** — Signaldaten werden gesammelt (0–116 s im Zeitschlitz).
- **Bernsteinfarbener Balken pulsiert** — Dekodierung läuft (letzte ~4 s des Zeitschlitzes).
- **Leerer Balken** — Warten auf die nächste gerade UTC-Minute.

Jeder erfolgreich dekodierte Spot erscheint in einer Tabelle mit folgenden Spalten:

Spalte	Bedeutung
UTC	Zeit des Spots (gerade Minute)
Callsign	Die Station, die gesendet hat
Grid	Maidenhead-Locator des Senders
Power	Sendeleistung in dBm
Freq	Genaue Audiofrequenz (Hz) innerhalb des WSPR-Durchlassbereichs
SNR	Signal-Rausch-Verhältnis in dB

Klicken Sie auf **Clear**, um die Spot-Liste zu löschen.

Tipp: Die WSPR-Dekodierung erfordert eine sehr genaue Systemzeit (innerhalb ± 1 Sekunde zu UTC). Der erste Zeitschlitz nach dem Aktivieren des Decoders beginnt mit der nächsten geraden UTC-Minute — eine kurze Wartezeit ist normal.

8. HF-FAX / WEFAX

Worum es geht: HF-Radiofax (auch WEFAX genannt) wird von Küstenwachen und Wetterdiensten weltweit genutzt, um Wetterkarten, Seegangskarten und Bodenanalysen über Kurzwelle auszustrahlen. Der Decoder setzt das Bild Zeile für Zeile zusammen, während es empfangen wird.

Einrichtung

1. Aktivieren Sie den Decoder und wählen Sie **HF FAX / WEFAX** aus dem Menü. Oder drücken Sie einfach die Taste **FAX**.
2. Das Bedienfeld **HF FAX / WEFAX Receiver** erscheint darunter.
3. **Wählen Sie eine Station** im Auswahlmenü Station. Über 20 Stationen stehen zur Verfügung und decken Europa, Asien, Ozeanien und Amerika ab (z. B. DDH3/DDK3 Deutschland, SVJ4/GR Griechenland, JMH Japan, NMG USA New Orleans). Auch CBM Punta Arenas (Chile) ist dabei. Die Sendepäne folgen dem weltweiten Radiofax-Plan der NOAA (Ausgabe vom 7. März 2025); Charleville (VMC, Australien) enthält seine zusätzlichen Sendungen, die nur auf 11030 kHz laufen.
4. Sendet die Station auf mehreren Frequenzen, wählen Sie die gewünschte im Untermenü **Frequency**.
5. Klicken Sie auf ► **Tune**, um den Wasserfall automatisch auf diese Station abzustimmen.

6. Die Betriebsart wird automatisch auf **USB** gesetzt.

Sendeplan

Ist eine Station gewählt, erscheint eine Countdown-Tabelle **Next Transmissions** mit den nächsten 4 geplanten Aussendungen in UTC und einem laufenden Countdown:

- Normal (grau/grün) — die Aussendung steht bevor.
- **Bernstein** — die Aussendung beginnt in weniger als 3 Minuten; jetzt den Decoder scharfschalten.
- **Rot** ● — die Aussendung beginnt in weniger als 30 Sekunden; der Empfang steht unmittelbar bevor.

Parameter

Die meisten Stationen verwenden die Standardvorgaben (mit ★ markiert). Ändern Sie sie nur, wenn Sie wissen, dass die Station abweichende Einstellungen nutzt.

Parameter	Vorgabe	Beschreibung
LPM	120 ★	Zeilen pro Minute — die Trommeldrehzahl
IOC	576 ★	Kooperationsindex — bestimmt die Pixel pro Zeile
Shift	800 Hz ★	Frequenzhub zwischen Schwarz- und Weißton

Bedienelemente

- ⇌ **Auto-align** — standardmäßig aktiviert. Synchronisiert automatisch auf das Phasensignal zu Beginn jedes Bildes. Deaktivieren Sie es nur, wenn Sie bei einem nachweislich guten Signal Ausrichtungsprobleme haben.
- ⬆️ **Invert** — vertauscht Schwarz und Weiß. Verwenden Sie es, wenn das Bild als Negativ erscheint (weiße Flächen dort, wo Schwarz sein sollte).
- ↺ **Refresh** — löscht die Zeichenfläche und setzt den Decoder zurück. Nutzen Sie es zwischen zwei Aussendungen oder wenn das Bild reißt oder verläuft.
- ⬇️ **Save PNG** — speichert die aktuelle Zeichenfläche als PNG-Datei auf Ihrem Rechner.

Statusanzeigen

Unter dem Bild zeigen zwei Tonanzeigen:

- **300 Hz phasing** — leuchtet cyan, wenn der Phasenton am Bildanfang erkannt wird.
- **450 Hz stop** — leuchtet rot, wenn der Stoppton am Bildende erkannt wird.

Beide Signale werden an ihrem Rhythmus über die ganze Zeilenbreite erkannt, die echte Start- und Stoppsignale immer ausfüllen und Karteninhalte praktisch nie, sodass schraffierte Flächen einer Karte nicht für ein Startsignal gehalten werden. Sie werden bis herab zu etwa 0 dB SNR erkannt, und bei sehr schwachen Signalen wird das Bild aus dem Mittel mehrerer Phasing-Zeilen ausgerichtet, sodass die Karte trotzdem am linken Rand beginnt.

Hinweis: Das Bild läuft nach oben — die zuletzt empfangene Zeile erscheint immer unten auf der Zeichenfläche. Steht **[PHASING]** in der Kopfzeile, hat der Decoder auf einen neuen Bildanfang eingerastet.

9. NAVTEX

Worum es geht: NAVTEX ist das internationale seefunkgestützte Aussendungssystem für küstennahe Sicherheitsinformationen — Navigationswarnungen, Wettervorhersagen sowie Such- und Rettungsmeldungen. Es verwendet 100-Baud-FSK (SITOR-B mit FEC) und wird weltweit auf eigenen Kanälen empfangen.

Verfügbare Kanäle

Kanal	Frequenz	Verwendung
International	518 kHz	Englischsprachig, international
National	490 kHz	Aussendungen in der Landessprache
KW (×5)	4209.5 / 6314 / 8416.5 / 12579 / 16806.5 kHz	KW-NAVTEX mit großer Reichweite

Einrichtung

1. Aktivieren Sie den Decoder und wählen Sie **NAVTEX** aus dem Menü. Oder drücken Sie einfach die Taste **NAVTEX**. Der Empfänger schaltet automatisch auf **USB**.
2. Das Bedienfeld **NAVTEX Receiver** erscheint.
3. Wählen Sie den gewünschten Kanal im Menü **Station** (z. B. **International – 518 kHz**).
4. Klicken Sie auf ⇒ **Tune & Set IF**, um den Wasserfall automatisch abzustimmen und den Durchlassbereich auf das richtige Audiofenster einzuengen. Die Betriebsart wird automatisch auf **USB** gesetzt. Die Anzeigefrequenz liegt 500 Hz unter der Kanalmitte, sodass das NAVTEX-Signal bei 500 Hz im Audio erscheint.

Sendeplan

Die Plantabelle listet alle bekannten Stationen des gewählten Kanals mit:

- ihrem ITU-Kennbuchstaben und der Landesflagge
- der nächsten Sendezeit in UTC
- einem laufenden Countdown bis zur nächsten Aussendung
- **Bernstein** bei unter 2 Minuten / **Rot** ● bei unter 30 Sekunden — jetzt den Decoder scharfschalten

Die Ausgabe lesen

Der dekodierte Text erscheint in blaugrüner Festbreitenschrift. Die Meldungsgrenzen sind klar markiert:

```
— ZCZC MA12 —
... message content ...
— NNNN —
```

ZCZC markiert den Anfang einer Meldung. Die vier folgenden Zeichen kennzeichnen Station (**M**), Thema (**A** = Navigationswarnungen) und laufende Nummer (**12**). Das Banner wartet auf die vollständige Kopfzeile und lautet daher **ZCZC MA12**, nicht **ZCZC ????**. **NNNN** markiert das Ende.

Stimmen Sie ein, nachdem eine Meldung schon begonnen hat, oder geht ihre Kopfzeile im Rauschen unter, erscheint der Text trotzdem, unter einem Banner — **(header missed)** —. Es öffnet sich erst, wenn mehrere Zeichen hintereinander eintreffen, bei denen beide NAVTEX-Aussendungen übereinstimmen, was Rauschen praktisch nie erzeugt. Es schließt wie gewohnt bei **NNNN**, mit — **(signal lost)** —, wenn das Signal vorher verschwindet, oder wenn eine neue Meldung beginnt.

Klicken Sie auf **Clear**, um den Meldungspuffer zu löschen.

Hinweis: NAVTEX arbeitet auf MF- und LF-Frequenzen (518/490 kHz). Die Empfangsreichweite beträgt typischerweise 200–400 Seemeilen vom Sender. Die KW-Kanäle (4–17 MHz) bieten eine deutlich größere Reichweite.

10. FSK / RTTY — einschließlich PSK31 und Olivia

Worum es geht: Ein universeller Decoder für schmalbandige Textbetriebsarten mit fünf Varianten, die über ein einziges Auswahlmenü erreichbar sind. Drei davon sind echtes FSK (Frequency-Shift Keying): maritimes FSK (SITOR), Wetter-RTTY und Amateur-RTTY. Die anderen beiden sind überhaupt kein FSK, teilen sich aber dasselbe Fenster: **PSK31**, eine Phasenumtastung, und **Olivia**, ein Mehrton-FSK mit Vorwärtsfehlerkorrektur. Jede Variante bringt eine auf ihre Standardparameter abgestimmte Voreinstellung mit.

Varianten und Voreinstellungen

Variante	Mitte	Shift	Baud	Rahmen	Kodierung
Maritimes FSK / SITOR	500 Hz	170 Hz	100	7N1	CCIR-476
Wetter-RTTY	1000 Hz	450 Hz	50	5N1.5	ITA2
Amateur-RTTY	1000 Hz	170 Hz	45.45	5N1.5	ITA2
PSK31 (BPSK)	1000 Hz	—	31.25	—	Varicode
Olivia (MFSK)	1000 Hz	—	siehe Mode	—	7 Bit + FEC

Das Bedienfeld passt sich der gewählten Variante an. **Shift, Baud, Framing, Encoding, Invert mark / space** und **Auto shift detect** werden bei PSK31 und Olivia ausgeblendet, da keine der beiden Betriebsarten ein Mark/Space-Tonpaar oder eine UART-artige Rahmung besitzt. Stattdessen wird **Center audio** zu einem frei beschreibbaren Zahlenfeld (der Träger kann überall im Durchlassbereich liegen), und Olivia erhält zusätzlich einen **Mode**-Wähler und einen **Squelch**-Regler.

Einrichtung

1. Aktivieren Sie den Decoder und wählen Sie **FSK / RTTY** aus dem Menü. Oder drücken Sie einfach die Schaltfläche **RTTY**.
2. Das Decoder-Bedienfeld erscheint. Sein Titel richtet sich nach der Variante — *FSK / RTTY Decoder*, *PSK31 Decoder* oder *Olivia Decoder*.
3. Wählen Sie die **Variante**. Die Parameter werden automatisch aktualisiert.
4. Stellen Sie bei Olivia den **Mode** (Töne / Bandbreite) passend zur Aussendung ein — siehe die Olivia-Hinweise weiter unten.
5. Wählen Sie im Menü **Known frequency** eine gebräuchliche Frequenz für die gewählte Variante und klicken Sie dann auf **Tune**, um dorthin zu springen.
6. Feinabstimmen, bis der dekodierte Text stabil und lesbar wird.

Bekannte Frequenzen nach Variante

Maritimes FSK / SITOR

- 518,0 kHz — internationales NAVTEX
- 490,0 kHz — nationales NAVTEX
- 4209,5 / 6314,0 / 8416,5 / 12579,0 / 16806,5 / 22376,0 kHz — HF-SITOR

Wetter-RTTY

- 4583,0 / 7646,0 / 10100,8 / 11039,0 / 14467,3 kHz — DWD (Deutscher Wetterdienst)

Amateur-RTTY

- 3590 kHz (80 m), 7043 kHz (40 m), 10143 kHz (30 m), 14083 kHz (20 m), 21083 kHz (15 m), 28083 kHz (10 m)

PSK31

- 3580,15 kHz (80 m), 7040,15 kHz (40 m), 10142,15 kHz (30 m), 14070,15 kHz (20 m), 18100,15 kHz (17 m), 21080,15 kHz (15 m), 24920,15 kHz (12 m), 28120,15 kHz (10 m)

Olivia

- 3577,75 kHz (80 m), 7073,75 kHz (40 m), 10142,25 kHz (30 m), 14075,5 kHz (20 m), 18103,75 kHz (17 m), 21075,75 kHz (15 m), 24921,75 kHz (12 m), 28123,75 kHz (10 m)

Parameter

Parameter	Gilt für	Beschreibung
Center audio (Hz)	alle	Die Audiofrequenz der Mitte zwischen Mark und Space; bei PSK31 der Träger, bei Olivia die Mitte des Tonblocks. Ein Auswahlmenü bei den FSK-Varianten, ein freies Eingabefeld bei PSK31 und Olivia
Shift (Hz)	nur FSK	Frequenzunterschied zwischen Mark- und Space-Ton
Baud	nur FSK	Symbolrate
Framing	nur FSK	Datenbits, Parität, Stoppbits (z. B. 7N1 = 7 Daten, keine Parität, 1 Stopp)
Encoding	nur FSK	Zeichensatz (CCIR-476, ITA2/Baudot oder ASCII)
Invert mark / space	nur FSK	Vertauscht Mark- und Space-Ton
Auto shift detect	nur FSK	Versucht, den Shift automatisch aus dem eingehenden Signal zu messen
Mode (Töne / Hz)	nur Olivia	Anzahl der Töne und Bandbreite — muss exakt zur Aussendung passen
Squelch (FEC S/N)	nur Olivia	Wie stark die Übereinstimmung der Fehlerkorrektur sein muss, bevor Text ausgegeben wird
Squelch (SNR)	Amateur- und Wetter-RTTY	Verhindert, dass Rauschen als zufällige Buchstaben ausgegeben wird. Beginnt bei –5 dB für Amateur-RTTY und –8 dB für Wetter-RTTY; ganz links (off) ist sie abgeschaltet. Die SNR-Anzeige wird grau, solange die Squelch geschlossen ist

Signalmesswerte

Die Statuszeile zeigt Live-Messwerte, und die Felder wechseln mit der Variante:

Variante	Angezeigte Felder
FSK-Varianten	Mark / Space (gemessene Tonfrequenzen), SNR , Lock , Timing
PSK31	Carrier (Hz, nach automatischer Frequenzkorrektur), IMD (dB), SNR , Lock , Timing
Olivia	Centre (Hz), Mode , S/N , FEC (%), Sync

Timing/Sync zeigt LOCKED/SYNCED, sobald der Decoder dem Signal folgt, und SEARCH, solange er noch sucht.

Die **SNR** der FSK-Varianten und von PSK31 wird am Signal selbst gemessen und auf 3 kHz Bandbreite bezogen, sodass beide auf derselben Skala liegen: von +20 dB bis –5 dB auf etwa 1 dB genau. Eine leere Frequenz zeigt etwa –12 bis –15 dB, und bei PSK31 ist 0 dB bereits ein starkes Signal. Die **S/N** von Olivia ist etwas anderes: die Qualitätszahl der Fehlerkorrektur.

Amateur-RTTY (45,45 Baud, 170 Hz) wird bis herab zu etwa 0 dB fehlerfrei gelesen, mit rund 1,5 % falschen Zeichen bei –3 dB und 20 % bei –6 dB.

Weitere Bedienelemente

- ⇒ **Set IF Band-Pass** — verengt den Durchlassbereich des Empfängers so, dass er das Signal eng umschließt. Die Breite richtet sich nach der Variante: 300 Hz für Amateur-RTTY (schmal genug, um eine stärkere Station 250 Hz daneben zu entfernen, und dennoch lesbar bei einem 40 Hz verstimmt Signal), 594 Hz für Wetter-RTTY, Mark/Space plus Reserve für Maritime FSK, etwa ± 100 Hz für PSK31 und die volle Breite des Tonblocks plus Reserve für Olivia. Der Durchlassbereich folgt dem Decoder, wenn **Auto-tune** oder **Auto shift detect** die Töne verschiebt, und wenn Sie die **Baud**-Einstellung ändern.
-  **Auto-tune Center** — automatische Suche nach dem Signal. Bei den FSK-Varianten sucht sie ein ausgewogenes Tonpaar, bei PSK31 findet sie den Träger, bei Olivia den stärksten Block der gewählten Bandbreite. **Auto shift detect** ändert die Shift nur, wenn ein echtes Zweitonsignal vorhanden ist, und springt daher nicht mehr auf Rauschen.

Hinweise zu PSK31

PSK31 ist die verbreitetste Tastatur-zu-Tastatur-Betriebsart auf Kurzwelle. Sie ist nur 62 Hz breit, sodass mehrere Verbindungen innerhalb weniger hundert Hertz um die übliche Sammelfrequenz nebeneinander liegen; das Abstimmen besteht darin, im Wasserfall eine Spur aus der Gruppe herauszugreifen.

- Der Decoder korrigiert seinen Abstimmfehler über etwa **± 25 Hz** selbst, Sie müssen also nur in die Nähe kommen. **Carrier** in der Messwertzeile zeigt, wo er sich tatsächlich eingependelt hat.
- **IMD** misst die Senderqualität, nicht den Empfang: ein sauberes Signal liegt bei etwa –20 dB oder besser. Ein schlechter Wert bedeutet, dass die Gegenstation übersteuert, nicht dass Sie falsch abgestimmt sind.
- Der Text erscheint Zeichen für Zeichen ohne Fehlerkorrektur; ein schwaches Signal führt daher zu vereinzelt falschen Buchstaben, statt ganz abzubrechen.

Hinweise zu Olivia

Olivia tauscht Geschwindigkeit gegen Robustheit. Es ist deutlich langsamer als PSK31, dekodiert aber Signale, die mit dem Ohr nicht mehr hörbar sind — daher seine Beliebtheit für schwache Signale und große Entfernungen.

- **Die Mode-Einstellung muss exakt zur Aussendung passen.** Eine falsche Kombination aus Tönen und Bandbreite dekodiert überhaupt nichts — keinen verstümmelten Text, sondern Stille. Das Bedienfeld startet mit **8 / 250**, der schmalen Konfiguration, die auf den Anruffrequenzen mitläuft; **16 / 500** und **32 / 1000** sind die beiden anderen gebräuchlichen, und **16 / 1000** wird ebenfalls angeboten.
- Olivia sendet keine Präambel, der Decoder muss die Synchronisierung daher suchen. **Rechnen Sie nach dem Abstimmen mit einigen Sekunden**, bevor Text erscheint. Das Feld **Sync** zeigt bis zum Einrasten SEARCH.
- Die Fehlerkorrektur arbeitet blockweise, der Text trifft daher **schubweise statt gleichmäßig** ein, mit einer Verzögerung von mehreren Blöcken zwischen Aussendung und Anzeige.
- **Squelch (FEC S/N)** legt fest, wie sicher sich die Fehlerkorrektur sein muss, bevor sie ausgibt. Die Voreinstellung 4,0 hält Rauschen fern; 3,0 ist die Untergrenze, unterhalb derer zufälliges Rauschen

vereinzelt Zeichen ausgibt. Ein gutes Signal erreicht 8–9 auf der **FEC**-Anzeige, es bleibt also reichlich Spielraum, den Squelch auf einem belegten Band anzuheben.

Hinweis zur Betriebsart: Der Decoder übernimmt die Kontrolle über Demodulationsart und ZF-Durchlassbereich, solange er aktiv ist. Beides wird beim Abschalten automatisch wiederhergestellt. Alle fünf Varianten verwenden **USB**.

Hinweis zur Polarität (nur FSK-Varianten): Bei Wetter-RTTY muss meist **Invert mark / space** aktiviert werden. Bei maritimem FSK (SITOR/NAVTEX-Art) und Amateur-RTTY lassen Sie es deaktiviert — Amateur-RTTY sendet Mark als die höhere Hochfrequenz, und USB behält sie als den höheren Audioton bei, was genau dem deaktivierten Fall entspricht. Ist der dekodierte Text verstümmelt, sollten Sie zuerst dieses Kästchen umschalten. Bei PSK31 und Olivia ist es ausgeblendet, da sie kein Mark/Space-Paar besitzen.

Hinweis zu Buchstaben/Ziffern (nur FSK-Varianten): Baudot führt Buchstaben und Ziffern in zwei getrennten Zuständen, und Rauschen kann den Decoder in den falschen kippen — wodurch nicht nur das gestörte, sondern jedes folgende Zeichen verstümmelt wird. Der Decoder kehrt deshalb bei jedem Leerzeichen zu Buchstaben zurück; das ist übliche Praxis und repariert eine verfälschte Umschaltung innerhalb von ein bis zwei Wörtern statt einer ganzen Zeile. Der Preis: Zifferngruppen, die durch Leerzeichen getrennt sind, erfordern, dass die Gegenstelle die Ziffernumschaltung nach jedem Leerzeichen wiederholt — was Sender normalerweise tun.

11. SSTV

Worum es geht: Slow-Scan-Television überträgt Standbilder über einen gewöhnlichen SSB-Sprechfunkkanal, Zeile für Zeile, als frequenzmodulierten Ton zwischen 1500 Hz (Schwarz) und 2300 Hz (Weiß). Ein vollständiges Bild dauert je nach Betriebsart zwischen 36 Sekunden und 4½ Minuten.

Empfohlene Frequenzen (USB)

Band	Frequenz	Betriebsart	Anmerkungen
20 m	14.230 MHz	USB	Die wichtigste internationale SSTV-Anruffrequenz — mit Abstand die aktivste
20 m	14.233 MHz	USB	Ausweichfrequenz, wenn 14.230 belegt ist
15 m	21.340 MHz	USB	
10 m	28.680 MHz	USB	Aktiv bei Bandöffnungen
40 m	7.171 MHz	LSB	
80 m	3.845 MHz	LSB	Regional, abends

Das Seitenband wird für Sie gewählt. Beim Start des Decoders wird **LSB unterhalb 10 MHz** und **USB oberhalb** gewählt, entsprechend der üblichen Amateurfunkpraxis. Auf dem falschen Seitenband ist die Tonzuordnung invertiert, und das Bild lässt sich nicht dekodieren. Treffen Sie auf eine Station, die sich nicht an die Konvention hält, wechseln Sie das Seitenband einfach von Hand — der Decoder läuft weiter, löscht das Bild und beginnt mit der neuen Einstellung von vorn.

Einrichtung

1. Aktivieren Sie den Decoder und wählen Sie **SSTV** aus dem Menü. Oder drücken Sie einfach die Taste **SSTV**. Der Empfänger wechselt das Seitenband automatisch — USB über 10 MHz, LSB darunter.

2. Stimmen Sie so ab, dass die Bildtöne in der Mitte des Durchlassbereichs liegen. Ein richtig abgestimmtes Signal hat seine Synchronimpulse bei 1200 Hz und den Bildinhalt zwischen 1500 und 2300 Hz.
3. Belassen Sie **Mode** auf **Auto**, sofern Sie nicht bereits wissen, was gesendet wird. Das Bild baut sich Zeile für Zeile auf.

Betriebsarten

Einstellung	Bild	Dauer
Auto	automatisch erkannt	—
Martin M1 / M2	320×256 Farbe	114 s / 58 s
Scottie S1 / S2	320×256 Farbe	110 s / 71 s
Scottie DX	320×256 Farbe	269 s
Robot 36 / 72	320×240 Farbe	36 s / 72 s

Auto arbeitet auf zwei Wegen: Es liest den **VIS-Kopf** — den digitalen Betriebsartencode, der in den ersten 300 ms einer Aussendung gesendet wird — und bestimmt, falls der Kopf verpasst wurde (Sie sind zu spät eingestiegen oder er ging im QSB verloren), die Betriebsart stattdessen aus dem Timing der Synchronimpulse. Die Wahl einer bestimmten Betriebsart erzwingt diese, aber der Decoder prüft dennoch die Synchronisation, bevor er etwas zeichnet — eine falsche Wahl liefert also kein Bild statt Rauschen.

Die Ausgabe lesen

Die Statuszeile unter den Bedienelementen zeigt, was der Decoder gerade tut:

Status	Bedeutung
Waiting for VIS / AUTO lock	Horcht; noch nichts erkannt
Martin M1? verifying sync...	Ein Kandidat wurde gefunden und wird anhand der nächsten Zeilen bestätigt
Martin M1 lock (VIS)	Eingerastet über den VIS-Kopf
Martin M1 lock (AUTO)	Eingerastet über das Synchron-Timing
Martin M1 lock (MANUAL)	Über die Schaltfläche ● Force gestartet
Candidate rejected (no sync)	Der Kandidat wurde nicht bestätigt — bei Rauschen normal
– sync lost, resetting	Das Signal verschwand mitten im Bild
Frame complete	Das vollständige Bild wurde empfangen

Die erkannte Betriebsart erscheint außerdem grün neben dem Titel **SSTV**, und der Zeilenzähler zeigt den Fortschritt.

Bedienelemente

Schaltfläche	Aktion
► Start	Schaltet den Decoder scharf. Er zeichnet nicht von selbst ein Bild — ein Einrasten muss weiterhin über den VIS-Kopf oder die Synchronerkennung erfolgen.
■ Stop	Beendet die Dekodierung.
↺ Reset	Löscht die Zeichenfläche und schaltet an Ort und Stelle wieder scharf. Zwischen zwei Bildern oder nach dem Nachstimmen zu verwenden.
● Force	Beginnt sofort in der im Menü gewählten Betriebsart zu zeichnen und überspringt VIS- und Synchronerkennung vollständig. Nur verfügbar, wenn Mode nicht auf Auto steht.
Save	Speichert das aktuelle Bild als PNG.

Wann Force zu verwenden ist. Wenn Sie im Wasserfall ein Bild sehen, der Decoder aber nicht darauf einrastet — eine ungewöhnliche Betriebsart, ein für die Detektoren zu schwaches oder zu verzerrtes Signal oder ein verpasster Kopf —, wählen Sie die Betriebsart von Hand und drücken Sie ● **Force**. Das Bild wird im Moment des Klicks verankert, und die Betriebsartenmarkierung zeigt **MANUAL**, sodass Sie es von einem VIS- oder AUTO-Einrasten unterscheiden können. Der Decoder rastet dennoch auf einen echten Synchronimpuls ein, wenn er einen findet — ein etwas zu früher oder zu später Klick wird also korrigiert.

Da Force sämtliche Sicherheitsprüfungen überspringt, malt es bereitwillig Rauschen, wenn Sie es auf einem leeren Kanal drücken, und es stoppt nicht von allein — drücken Sie ■ **Stop** oder ↺ **Reset**.

Anmerkungen

- **Rauschen startet den Decoder nicht.** Statische Entladungen, Funkenstörungen und Netzbrumm reichten früher aus, um eine Dekodierung auszulösen. Jede Erkennungsstufe prüft nun, dass der Ton wirklich ein Ton ist, und bestätigt den Kandidaten anhand der folgenden Zeilen, bevor ein einziger Bildpunkt gezeichnet wird. Erwarten Sie, dass der Decoder auf einem leeren Band still bleibt.
- **Das Bild verläuft schräg, wenn Sie neben der Frequenz liegen.** SSTV verzeiht Abstimmfehler nicht. Neigen sich die Zeilen, stimmen Sie in kleinen Schritten nach und lassen Sie das nächste Bild neu beginnen.
- Farbbalance und Schärfe sind fest eingestellt; es gibt nichts nachzuregeln.

12. Allgemeine Tipps

Decoder: ON muss zuerst eingeschaltet werden. Das Auswahlmenü ist deaktiviert (ausgegraut), bis Sie die Decoder-Schaltfläche anklicken.

Immer nur ein Decoder. Die Auswahl eines neuen Decoders im Menü beendet automatisch den zuvor aktiven und setzt alle von ihm vorgenommenen Betriebsart- oder Durchlassbereichsänderungen zurück.

Die Betriebsart wird für Sie verwaltet. HF-FAX und NAVTEX schalten den Empfänger sofort nach der Auswahl auf USB, SSTV wählt das Seitenband anhand des Bandes (LSB unter 10 MHz, USB darüber), und FAX, NAVTEX sowie FSK passen zusätzlich den Durchlassbereich an, wenn Sie ihre Tune-Schaltfläche anklicken. FT8, FT4, FT2, JS8 und WSPR tun dasselbe, sobald Sie sie auswählen — über die Schaltflächenreihe oder das Auswahlmenü: Der Empfänger schaltet auf USB und erhält den passenden Durchlassbereich — das gesamte 3-kHz-Teilband für die FT8-Familie, 1350–1650 Hz für WSPR. Beim Beenden des Decoders wird die Standardbetriebsart des Bandes wiederhergestellt.

Das Einschalten eines Decoders unterbricht den Ton nicht mehr. Die Decoder laufen in eigenen Threads, sodass es beim Starten, Stoppen oder Wechseln weder Lücke noch Klick noch Aussetzer gibt.

Die Genauigkeit der Systemuhr zählt. FT8, FT4 und WSPR sind zeitkritisch. Sie dekodieren in festen, an UTC ausgerichteten Fenstern. Weicht Ihre Rechneruhr um mehr als 1–2 Sekunden ab, sinken die Dekoderraten deutlich. Halten Sie Ihre Uhr mit einem NTP-Client genau.

Die Rauschfilter erreichen die Decoder nicht. NR, NB, NS und AN sind Hörhilfen, ausschließlich für Ihre Ohren. Jeder Decoder — FT8, FT4/FT2, CW, WSPR, FAX, NAVTEX, FSK/RTTY/PSK31/Olivia, SSTV und der QRSS-Grabber — greift das Audio *vor* diesen Filtern ab; stellen Sie sie also so ein, wie es am besten klingt, ohne sich um die Decodierqualität zu sorgen. Aus demselben Grund laufen die Decoder weiter, während der Empfänger stummgeschaltet oder zugerauscht ist: Sie können den Lautsprecher abschalten und einen Decoder oder eine Nacht-QRSS-Aufzeichnung weiterlaufen lassen. Was dem Gehörten tatsächlich folgt, ist allein das Audio-Spektrogramm, das bewusst das gefilterte Audio zeigt.

Signalqualität schlägt Signalstärke. Die meisten dieser Decoder sind für schwache Signale ausgelegt. Ein ruhigeres Band mit geringerem Rauschen ist oft ergiebiger als ein lautes, störungsreiches Signal. Nutzen Sie Wasserfall und Durchlassbereichseinstellungen, um QRM zu erkennen und zu meiden, bevor Sie einen Decoder aktivieren.

Nutzen Sie die Schaltflächen Refresh oder Clear großzügig. FAX-Bilder verlaufen, wenn die Anzeigefrequenz leicht danebenliegt, und Textdecoder sammeln Störzeichen an. Ein Neuanfang nach Abstimmkorrekturen liefert oft eine deutlich sauberere Ausgabe.

Automatisches Spot-Reporting und Systemgraphen

FT8-, FT4- und WSPR-Dekodierungen kann auch der Server selbst hochladen — FT8/FT4 an PSK Reporter, WSPR an WSPRnet — über einen Autorun-Daemon, den der Sysop im Admin-Panel startet. Er ist von den Decodern in Ihrem Browser unabhängig: er läuft weiter, ob jemand zuhört oder nicht, und nichts, was Sie im Browser dekodieren, wird gemeldet.

Der Sysop verfolgt ihn über zwei Zähler, die leicht zu verwechseln sind: Die Kacheln je Decoder zählen die seit dem letzten Start des Daemons hochgeladenen Spots, während die Zahl neben dem Kontrollkästchen jedes Bands bzw. jeder Betriebsart der Gesamtwert seit Beginn ist und Neustarts übersteht. Dasselbe Panel enthält eine Seite **Graphen**, die CPU-Takt, CPU-Last, CPU-Temperatur und Nutzer online über die letzten 15 Minuten bis 24 Stunden darstellt. Beides beschreibt das [Admin-Panel-Handbuch](#).

KAPITEL 9

RADE README

RADE_README.md

RADE v1 Digitalsprache für PhantomSDR-Plus

RADE (Radio AutoencoDEr) ist FreeDV's führende Betriebsart für digitale Sprache auf Kurzwelle. Sie nutzt einen Hybrid aus maschinellem Lernen und DSP (den neuronalen FARGAN-Vocoder) und liefert hochwertige Sprache über Kurzwelle bei Signal-Rausch-Abständen bis hinunter zu -2 dB, in nur 1500 Hz HF-Bandbreite — schmäler als ein SSB-Signal.

Dieses Dokument beschreibt die vollständige Integration des RADE-v1-Empfangs in PhantomSDR-Plus, umgesetzt als Python-Sidecar (`rade_helper.py`), der den Browser mit der Dekodierkette `radac_rxe.py` + `lpcnet_demo` verbindet.

Installation — der kurze Weg

```
cd ~/PhantomSDR-Plus
./install_rade.sh
```

Das ist die gesamte Installation. Das Skript installiert die Systempakete über den Paketmanager der Maschine (`apt`, `pacman`, `dnf` oder `zypper`), weicht für Module, die eine Distribution nicht mitbringt, auf `pip` aus, aktualisiert `websockets`, wenn die Distributionsversion älter als die vom Sidecar benötigte 11.0 ist, klonet und baut `radac`, prüft die Modellgewichte, baut das Frontend neu und startet den Sidecar. Unter Ubuntu 22.04 übergibt es für Sie an `install_rade_ubuntu22.sh`. `./install.sh` bietet an, es im Rahmen der normalen PhantomSDR-Plus-Installation auszuführen — auf einer frischen Maschine ist RADE also bereits vorhanden.

Dieses Dokument erklärt, wie RADE funktioniert und wie man es betreibt — es ist nicht die Installationsanleitung. Wenn Sie die Installation von Hand durchführen müssen, auf einem System, das das Skript nicht abdeckt, oder um einen einzelnen Schritt zu reparieren, folgen Sie der unten verlinkten Anleitung; nur dort sind die Schritte niedergeschrieben. Alles hier — die Architektur, die gepatchten Dateien, der Port, die Nutzung von RADE im Browser, die Umgebungsvariablen, die Fehlerbehebung und die Messung der Gleichzeitigkeit — gilt für jede Installation, gleich wie sie zustande kam.

- **Manuelle Installation unter Linux** - der Weg von Hand, Schritt für Schritt, für Ubuntu, Debian, Fedora, Arch und Raspberry Pi OS

Inhaltsverzeichnis

1. Installation — der kurze Weg
2. Funktionsweise
3. Architekturüberblick
4. RADEL gegenüber RADEU
5. Voraussetzungen
6. Installation von Hand
7. Die gepatchten Dateien ausbringen

8. Steuerung des Sidecars
 9. Port 8074
 10. RADE im Browser nutzen
 11. Überprüfung und Fehlersuche
 12. Umgebungsvariablen
 13. Geänderte Dateien
 14. Zusammenfassung des Signalflusses
 15. Fehlerbehebung
 16. Gleichzeitigkeit auf eigener Hardware messen
 17. RADE v1 aktualisieren
-

Funktionsweise

RADE v1 kann nicht im Browser laufen — es benötigt PyTorch und den neuronalen FARGAN-Vocoder, die für WASM zu groß sind. Die Lösung ist ein Python-Sidecar-Prozess (`rade_helper.py`), der auf demselben Server wie PhantomSDR-Plus läuft.

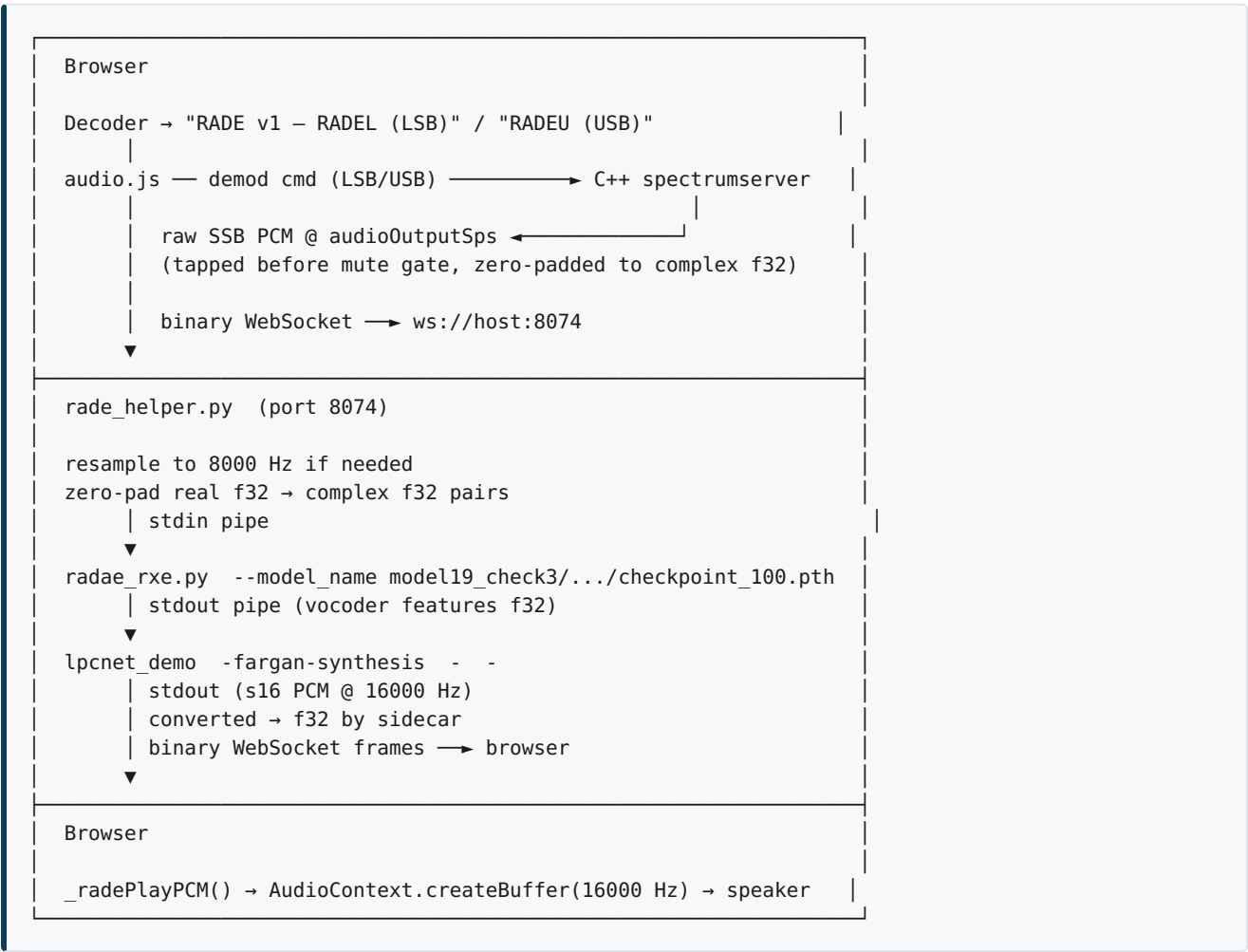
Wichtig: `freedv_rx` aus dem `codec2`-Repository unterstützt RADEV1 **nicht**. Die RADE-Dekodierkette liegt vollständig im separaten Repository `radae` von David Rowe (VK5DGR). Versuchen Sie nicht, `codec2s freedv_rx` für RADE zu verwenden.

Wenn Sie RADE im Browser auswählen:

1. Das Frontend stellt die zugrunde liegende Demodulation auf USB oder LSB — der C++-Server demoduliert das SSB-Signal wie gewohnt.
2. Das rohe demodulierte PCM wird in `audio.js` **vor** jeder Stummschaltung oder Rauschsperre abgegriffen — `radae_rxe.py` braucht einen kontinuierlichen Eingang, um die Rahmensynchronisation zu halten.
3. Jeder PCM-Block wird von reellem f32 auf komplexes f32 mit Nullen aufgefüllt (Realteil + 0.0 Imaginärteil) — genau das erwartet `radae_rxe.py` auf der Standardeingabe.
4. Der Sidecar leitet diese Abtastwerte an `radae_rxe.py` weiter, das Vocoder-Merkmale ausgibt.
5. `lpcnet_demo -fargan-synthesis` wandelt die Merkmale in s16-Sprache mit 16000 Hz um.
6. Der Sidecar wandelt s16 → f32 und sendet das Ergebnis als binäre WebSocket-Rahmen zurück an den Browser.
7. Der Browser spielt die dekodierte Sprache über die Web-Audio-API mit 16000 Hz ab.

Der C++-Server (`spectrumserver.cpp`) wird überhaupt nicht verändert.

Architekturüberblick



RADEL gegenüber RADEU

RADE v1 wird stets über SSB übertragen. Konventionell gilt:

Betriebsart	Seitenband	Zu verwenden auf den Bändern
RADEL	LSB	160 m, 80 m, 40 m (≤ 10 MHz)
RADEU	USB	20 m, 17 m, 15 m, 12 m, 10 m (> 10 MHz)

Voraussetzungen

Anforderung	Version	Anmerkungen
PhantomSDR-Plus	beliebig	mit Vite-/Svelte-Frontend
Linux	Ubuntu 24.04+ / Debian Bookworm+	getestet
Python	3.8+	für <code>rade_helper.py</code> und <code>radar_rxe.py</code>
radar-Repository	aktueller master	liefert <code>radar_rxe.py</code> und <code>lpcnet_demo</code>
cmake	3.10+	um <code>lpcnet_demo</code> aus radar zu bauen
PyTorch	2.0+	von <code>radar_rxe.py</code> benötigt
Node.js	16+	für <code>npm run build</code>
websockets (Python)	10-16+	<code>pip3 install websockets</code> — alle Versionen unterstützt
matplotlib	beliebig	von <code>radar_rxe.py</code> beim Import benötigt
numpy	1.23+	erforderlich; ermöglicht auch Resampling
scipy	beliebig	optional, ermöglicht genaues Resampling

Installation von Hand

`./install_rade.sh` erledigt die gesamte Installation. Wenn Sie es selbst tun wollen — auf einem System, das das Skript nicht abdeckt, oder um einen Schritt zu reparieren — steht das Verfahren in der **RADE v1 — Anleitung zur manuellen Installation**, und nur dort. Sie deckt Ubuntu, Debian, Fedora, Arch und Raspberry Pi OS in einem Durchgang ab:

Schritt 1	Systempakete und die Prüfung auf Node.js 22+
Schritt 2	Python-Pakete, einschließlich der PEP-668-Option und des CPU-only-torch-Wheels
Schritt 3	Das radar-Repository klonen und bauen, <code>lpcnet_demo</code> und die Modellgewichte prüfen
Schritt 4	Die Dekodierkette offline überprüfen, bevor sie an den Browser angeschlossen wird
Schritt 5	Das PhantomSDR-Plus-Frontend bauen
Schritt 6	Steuerung des Sidecars
Schritt 7	Port 8074 öffnen
Schritte 8-10	Server starten, RADE überprüfen, im Browser nutzen

Der Rest dieses Dokuments setzt das als erledigt voraus.

Die gepatchten Dateien ausbringen

Kopieren Sie die folgenden Dateien aus dem Patch-Satz in Ihr PhantomSDR-Plus-Repository.

Neue Datei — in das Wurzelverzeichnis des Repositories legen, neben das von Ihnen genutzte Startskript:

```
rade_helper.py
```

Gepatchte Frontend-Dateien — nach `frontend/src/` legen:

```
audio.js
App.svelte
```

Was die Patches bewirken

audio.js — 5 Änderungen:

- Konstruktor: 5 neue RADE-Zustandsfelder (`decodeRADE`, `_radeSideband`, `_radeSocket`, `_radeCallback`, `_radeReady`, `_radeNextTime`)
- Speichern des PCM vor der Anhebung: `pcmArrayPreBoost` wird vor der 300-fachen FLAC-Anhebung gesichert, damit RADE die ursprüngliche Amplitude erhält (die Anhebung würde `radet_rxe.py` übersteuern)
- `playAudio()`: RADE-PCM-Abgriff mit dem Audio vor der Anhebung, vor Stumm-/Rauschsperr
- `playAudio()`: Wächter `if (this.decodeRADE) return` — unterdrückt die rohe SSB-Wiedergabe, während die von RADE dekodierte Sprache läuft
- Neue Methoden: `setRADEDecoding()`, `setRADECallback()`, `_radePlayPCM()` mit lückenloser, geplanter Wiedergabe über die Uhr `_radeNextTime` (spielt mit **16000 Hz** — der Ausgaberate von `lpcnet_demo`)

Jede Svelte-Variante — 6 Änderungen:

- `demodulationDefaults`: `RADEL {type: 'LSB', offsets: [2200, -700]}`, `RADEU {type: 'USB', offsets: [-700, 2200]}` — der Durchlassbereich beginnt 700 Hz vom Träger und ist 1500 Hz breit
- Zustandsvariablen: `radeEnabled`, `radeConnected`, `radeSynced`, `radeSnr`, `_radeDeactivate()`
- `_radeDeactivate()`: stoppt den Decoder und **stellt die Standardbetriebsart des Bandes** aus `bands-config.js` wieder her — identisches Verhalten zum Abschalten von FAX, NAVTEX, FSK
- `_deactivateAll()`: RADE-Aufräumzeile
- `activateSelectedDecoder()`: Zweige `radel` und `radeu`
- Decoder-Auswahlmenü: zwei neue `<option>`-Einträge
- Statusfeld: Verbindungspunkt, Sync-/SNR-Status, Fehlerbanner

Startskripte

Die Startskripte des Empfängers (`start-rx888mk2.sh`, `start-airspyhf.sh`, `start-rtl.sh`, `start-rspla.sh`, `start-fobos-hf.sh`, `start-fobos.sh`, `start-hackrf.sh`) starten das Sidecar selbst, sobald der Server läuft, und `stop-websdr.sh` stoppt es zusammen mit dem Server. `rade.sh` bleibt für den eigenständigen Betrieb des Sidecars — siehe [Steuerung des Sidecars](#).

Steuerung des Sidecars

Normalerweise brauchen Sie kein eigenes Steuerskript für RADE. Die Startskripte des Empfängers (`start-rx888mk2.sh`, `start-airspyhf.sh`, `start-rtl.sh`, `start-rspla.sh`, `start-fobos-hf.sh`, `start-fobos.sh`, `start-hackrf.sh`) starten das Sidecar selbst, sobald der Server bestätigt läuft, starten es neu, wenn es sich beendet, und melden seinen Zustand in ihrer Abschlusszusammenfassung — ✓ `RADE sidecar running`, oder ein `!`-Hinweis, warum es nicht aktiviert wurde (nicht installiert,

RADE_ENABLED=0, oder kein python3). `./stop-websdr.sh` stoppt es zusammen mit dem Server. Um den Server ohne RADE zu betreiben, starten Sie ihn mit `RADE_ENABLED=0`.

`rade.sh` bleibt für den eigenständigen Betrieb erhalten — das Sidecar ohne den PhantomSDR-Plus-Launcher zu betreiben oder es allein zu testen:

```
chmod +x ~/PhantomSDR-Plus/rade.sh
cd ~/PhantomSDR-Plus
./rade.sh start      # start sidecar with self-restarting watchdog
./rade.sh stop       # stop sidecar + watchdog + all lpcnet_demo children
./rade.sh restart    # stop then start cleanly
./rade.sh status     # show running / stopped + process info
```

Die Sidecar-Aktivität wird nach `~/PhantomSDR-Plus/rade.log` protokolliert:

```
tail -f ~/PhantomSDR-Plus/rade.log
```

Mischen Sie beides nicht, während der Server läuft. Ist der Watchdog eines Startskripts aktiv, holt er das Sidecar innerhalb von etwa fünf Sekunden zurück, sodass `./rade.sh stop` wie ein Fehlschlag aussieht. Um RADE bei laufendem Server zu stoppen, starten Sie den Server entweder mit `RADE_ENABLED=0` oder halten Sie mit `./stop-websdr.sh` alles an.

Wenn Sie `rade.sh` eigenständig betreiben, **verwenden Sie immer `./rade.sh stop`** — ein bloßes `pkill -f rade_helper.py` schlägt seinen eigenen Watchdog nicht, der den Prozess binnen 3 Sekunden neu startet.

Port 8074

Das Sidecar lauscht auf TCP-Port **8074**, und der Browser verbindet sich direkt damit; der Port muss also von außen erreichbar sein. Wie man ihn öffnet — `ufw`, `firewalld`, `iptables`, die NAT-Regel im Router und der Nginx-Proxy für den Fall, dass ein Provider den Port ganz sperrt — steht in [Schritt 7 der manuellen Anleitung](#).

Zwei Dinge sind es wert, hier wiederholt zu werden, weil sie die meiste Zeit kosten:

```
# Is the sidecar listening?
ss -tlnp | grep 8074
```

Warnung vor NAT-Hairpin: Ein Test mit `curl` vom Server auf seinen eigenen öffentlichen Hostnamen liefert häufig `Connection refused`, auch wenn der Port offen ist — viele Router leiten den Verkehr nicht zurück. Testen Sie immer von einer Maschine außerhalb, oder nutzen Sie **`https://portchecker.co`**.

RADE im Browser nutzen

1. Öffnen Sie Ihre PhantomSDR-Plus-Weboberfläche
2. Aktive Stationen finden Sie unter qso.freedv.org
3. Stimmen Sie auf die Anzeigefrequenz der Station ab
4. Wählen Sie unter **Decoder Options:**
5. **RADE v1 — RADEL (LSB)** für 40 m / 80 m / 160 m

6. **RADE v1 — RADEU (USB)** für 20 m / 17 m / 15 m / 12 m / 10 m

7. Klicken Sie auf **Decoder: ON**

Die Tasten RADEL / RADEU (ein Druck)

Die Schritte 4 und 5 lassen sich durch einen einzigen Druck ersetzen. Ein Tastenpaar **RADEL / RADEU** steht an drei Stellen zur Verfügung:

- neben der Überschrift **Modes selector** im Hauptpanel;
- im Popup-Fenster **Modes**;
- im Popup-Fenster **Bands**.

Ein Druck wählt den Decoder aus, schaltet den Decoder ON, schließt das Popup, in dem Sie gedrückt haben, und holt das RADE-Panel in den sichtbaren Bereich. Die Taste leuchtet blau, solange RADE läuft.

Ein erneuter Druck auf dieselbe Taste schaltet RADE ab — das Panel schließt sich, und Betriebsart und Durchlassbereich werden wie unten beschrieben zurückgesetzt. Die Tasten, das Dropdown-Menü **Decoder Options** und die ON/OFF-Taste teilen sich denselben Zustand.

Zustände der Anzeige im Bedienfeld

Anzeige	Bedeutung
Rot — „Connecting to sidecar...“	Port 8074 nicht erreichbar oder Sidecar läuft nicht
Gelb — „Searching for signal...“	Sidecar verbunden, noch kein RADE-Rahmen erkannt
Grün — „Synced · SNR x.x dB“	Dekodierung läuft — Sprache ist zu hören

Wenn Sie den Decoder abschalten

Betriebsart und Durchlassbereich kehren automatisch zum korrekten Standard für die aktuelle Frequenz zurück, wie in `bands-config.js` festgelegt — genau wie bei FAX, NAVTEX, FSK.

Überprüfung und Fehlersuche

Läuft der Sidecar?

```
ps aux | grep rade_helper | grep -v grep
ss -tlnp | grep 8074
```

Aktivität live beobachten

```
tail -f ~/PhantomSDR-Plus/rade.log
```

Wenn sich ein Browser verbindet:

```
[RADE] client connected: x.x.x.x:XXXXX
[RADE] x.x.x.x:XXXXX sps=12000 sideband=LSB
[RADE] x.x.x.x:XXXXX spawning pipeline (torch_threads=1)
```

Schneller WebSocket-Rauchtest

```
python3 - &lt;&lt;&lt; 'EOF'
import asyncio, websockets, json

async def test():
    async with websockets.connect('ws://localhost:8074') as ws:
        await ws.send(json.dumps({'type': 'init', 'sps': 8000, 'sideband': 'LSB'}))
        print(await ws.recv()) # expect: {"type": "status", "connected": true}

asyncio.run(test())
EOF
```

Browserkonsole (F12)

[RADE] ► ENABLED LSB @ 12000 Hz → helper ws://localhost:8074

Gut — der WebSocket ist offen. Der Hz-Wert entspricht dem `audioOutputSps` Ihres Servers (typischerweise 8000–12000 Hz).

[RADE] sidecar socket error

Port 8074 ist nicht erreichbar — prüfen Sie Firewall und NAT-Regel des Routers.

Umgebungsvariablen

Variable	Standard	Beschreibung
<code>RADE_HELPER_PORT</code>	8074	TCP-Port, auf dem der Sidecar lauscht
<code>RADE_HELPER_HOST</code>	0.0.0.0	Bindeadresse (127.0.0.1 hinter einem Proxy)
<code>RADAE_DIR</code>	<code>~/rade</code>	Wurzel des rade-Repositories
<code>RADE_MODEL</code>	<code>RADAE_DIR/model19_check3/checkpoints/checkpoint_epoch_100.pth</code>	Modellgewichte
<code>LPCNET_DEMO</code>	<code>RADAE_DIR/build/src/lpcnet_demo</code>	Binärdatei <code>lpcnet_demo</code>
<code>RADE_AUXDATA</code>	1	Auf 0 setzen, um <code>--noauxdata</code> an <code>rade_rxe.py</code> zu übergeben
<code>RADE_TORCH_THREADS</code>	1	PyTorch-/OpenBLAS-Threads je <code>rade_rxe.py</code> -Instanz — begrenzt die CPU-Last
<code>RADE_PIN_CORES</code>	1	Bindet jede Dekodierkette an einen eigenen Kernsatz; 0 deaktiviert das Pinning
<code>RADE_CORES_PER_CLIENT</code>	2	Zugewiesene Kerne pro Client, wenn Pinning aktiv ist

Geänderte Dateien

Datei	Art	Anmerkungen
rade_helper.py	Neu	Python-Sidecar: WebSocket-Server + Dekodierkette aus zwei Prozessen
frontend/src/audio.js	Geändert	4 Patches
frontend/src/App.svelte	Geändert	6 Patches
rade.sh	Neu	Steuerskript für das RADE-Sidecar im eigenständigen Betrieb (start/stop/restart/status)
start-rx88mk2.sh	Geändert	startet, überwacht und meldet das Sidecar; ebenso start-airspyhf.sh, start-rtl.sh, start-rspla.sh, start-fobos-hf.sh, start-fobos.sh, start-hackrf.sh
stop-websdr.sh	Geändert	stoppt das Sidecar zusammen mit dem Server
spectrumserver.cpp	Unverändert	Keine C++-Änderungen erforderlich

Zusammenfassung des Signalflusses



Fehlerbehebung

Rotes Banner — „Sidecar not reachable“

```
# Is sidecar running?
ps aux | grep rade_helper | grep -v grep

# Start it manually for testing
python3 ~/PhantomSDR-Plus/rade_helper.py &

# Check port is open externally – use portchecker.co NOT curl from the server
# (curl from the server uses NAT loopback and gives false "Connection refused")
```

Port 8074 auf portchecker.co trotz iptables-Regel geschlossen

Möglicherweise filtert der Provider den Port, oder die NAT-Regel des Routers fehlt. Möglichkeiten:

1. Einen anderen Port versuchen: `RADE_HELPER_PORT=8080 python3 rade_helper.py`
2. Über Ihren bestehenden öffentlichen Port per Nginx proxen (siehe [Schritt 7 der manuellen Anleitung](#))

`radae_rxe.py: error: unrecognized arguments: model_path`

Der Modellpfad muss mit `--model_name` übergeben werden, nicht als Positionsargument:

```
# WRONG – positional argument
python3 radae_rxe.py model19_check3/checkpoints/checkpoint_epoch_100.pth

# CORRECT – named argument
python3 radae_rxe.py --model_name model19_check3/checkpoints/checkpoint_epoch_100.pth
```

`ModuleNotFoundError: No module named 'matplotlib'`

```
pip3 install matplotlib
```

`radae_rxe.py` importiert `matplotlib` bedingungslos am Dateianfang.

`ModuleNotFoundError: No module named 'torch'`

```
pip3 install torch
```

`lpcnet_demo: No such file or directory`

Der `radae`-Build wurde nicht abgeschlossen. Neu bauen:

```
cd ~/radae/build
cmake .. && make -j$(nproc)
ls src/lpcnet_demo      # should exist now
```

Fehler `size mismatch in inference.sh`

`model19_check3` benötigt `--auxdata`, wenn mit `inference.sh` kodiert wird:

```
./inference.sh model19_check3/checkpoints/checkpoint_epoch_100.pth \
wav/brian_g8sez.wav /dev/null \
--rate_Fs --pilots --pilot_eq --eq_ls --cp 0.004 \
--write_rx rx.f32 --auxdata # ← required for model19_check3
```

Hinweis: Beim **Dekodieren** mit `radea_rxe.py` ist `--auxdata` der Standard — übergeben Sie es dort nicht.

Gelbe Anzeige — „Searching for signal“ — rastet nie ein

- Bestätigen Sie das richtige Seitenband: RADEL für ≤ 10 MHz, RADEU für > 10 MHz
- Prüfen Sie auf qso.freedv.org, ob gerade eine Station sendet
- RADE v1 nutzt 30 Träger in 1500 Hz Bandbreite — es erscheint als kompakte Gruppe im Wasserfall
- Rechnen Sie mit bis zu 1,5 Sekunden für das Einrasten

underrun!!! von aplay beim Kettentest (Schritt 3)

Nur beim Offline-Test mit Dateien zu erwarten. `radea_rxe.py` verarbeitet langsamer als die Datei-Ein-/Ausgabe, wodurch der Audiopuffer leerläuft. Beim Live-Empfang tritt das nicht auf, weil der Browser den Ton in Echtzeit liefert.

ConnectionClosedError: received 1011 (internal error)

Der Sidecar hat die WebSocket-Verbindung angenommen, ist aber intern abgestürzt, bevor er antworten konnte. Ursache ist eine inkompatible Version von `rade_helper.py` — eine ältere Version versuchte, einen `asyncio.StreamReader` als Standardeingabe eines Unterprozesses zu übergeben, was still fehlschlägt und mit Fehler 1011 schließt.

Abhilfe: Ersetzen Sie `rade_helper.py` durch die aktuelle Version aus dem Patch-Satz. Die aktuelle Version nutzt `os.pipe()` für die Interprozess-Pipe und ist mit websockets 10.x bis 16.x+ kompatibel.

Mehrere gleichzeitige Nutzer

Jede Browserverbindung startet ihr eigenes, unabhängiges Paar aus `radea_rxe.py` + `lpcnet_demo`, sodass jeder Nutzer frei auf eine andere Frequenz abstimmen kann.

Standardmäßig nutzt `radea_rxe.py` **alle verfügbaren CPU-Kerne** für PyTorch-Matrixoperationen und verursacht damit rund 900 % CPU-Last pro Instanz. `rade_helper.py` begrenzt das mit zwei Optimierungen:

1. `OMP_NUM_THREADS=1` (sowie die MKL-/OpenBLAS-Entsprechungen) — begrenzt PyTorch auf 1 Thread
2. Mit numpy vektorisierte Audio-Konvertierungsfunktionen — beseitigen den Aufwand von Python-Schleifen

Ergebnis: Jede Instanz belegt grob **8-10 %** eines Kerns, genug für eine RADE-Dekodierung in Echtzeit. Falls Sie Tonaussetzer hören, erhöhen Sie auf 2 Threads:

```
RADE_TORCH_THREADS=2 ./start-rx888mk2.sh # or your receiver's launcher
# standalone: RADE_TORCH_THREADS=2 ./rade.sh restart
```

Gleichzeitige RADE-Nutzer	Ungefähre CPU-Last
1	~9 %
5	~45 %
10	~90 %
20	~180 %

Auf einem i5-12450H (12 Threads), auf dem spectrumserver ~42 % und rx888_stream ~11 % belegen, haben Sie rund **1000 % Reserve** — genug für **20 und mehr gleichzeitige** RADE-Nutzer, bevor die CPU zum Problem wird.

Die obige Tabelle unterstellt, dass die CPU-Kosten linear mit der Nutzerzahl wachsen. Die Messung sagt etwas anderes. Ein Lauf von `rade_loadtest.py` auf demselben i5-12450H zeigte bis etwa 24 Zuhörern eine CPU-Last je Zuhörer deutlich unter dieser Tabelle, darüber einen steilen Anstieg, wobei die Maschine bei **32 Zuhörern durch die Package-Temperatur** gestoppt wurde — nicht durch die CPU und nicht durch die Bandbreite. Behandeln Sie diese Zahlen nur als groben Anhaltspunkt für kleine Nutzerzahlen und messen Sie Ihre eigene Maschine: siehe [Gleichzeitigkeit auf eigener Hardware messen](#). Beachten Sie, dass der Lasttest die Kette mit Rauschen statt einem echten RADE-Signal speist; die absolute CPU-Last bei echtem Verkehr kann daher abweichen — verlässlich ist die *Form* der Kurve.

Gleichzeitigkeit auf eigener Hardware messen

Die obigen Zahlen stammen von einer bestimmten Maschine. `rade_loadtest.py` (im Repository-Wurzelverzeichnis) misst dasselbe auf **Ihrer** — es fährt synthetische RADE-Zuhörer in Stufen gegen den Sidecar hoch und meldet die letzte Zuhörerzahl, die Ihre Maschine gehalten hat, bevor etwas nachgab.

Nach jeder Stufe lässt es die Maschine zur Ruhe kommen und erfasst dann Package-Temperatur, Gesamt-CPU, verfügbaren Arbeitsspeicher, Swap und die summierte RSS aller `radae_rxe.py`- und `lpcnet_demo`-Prozesse. Es stoppt beim ersten **Knick** — je nachdem, welche Grenze zuerst überschritten wird — und gibt die letzte stabile Zahl aus.

Voraussetzungen

```
sudo apt install python3-websockets python3-psutil
```

Führen Sie es **auf dem SDR-Host** aus (es liest lokale Sensoren und den Prozessspeicher), während spectrumserver und der RADE-Sidecar bereits laufen:

```
cd ~/PhantomSDR-Plus
python3 rade_loadtest.py
```

Strg-C baut jederzeit alle Verbindungen sauber ab.

Vor dem ersten Lauf

Öffnen Sie die Datei und prüfen Sie die beiden Blöcke am Anfang:

- **CONFIG** — `WS_URL` steht standardmäßig auf `ws://127.0.0.1:8074`; ändern Sie es, wenn Sie den Sidecar von seinem Standardport weggelegt haben. `RADE_SPS` (12000) und `RADE_SIDE BAND` (USB) sollten dem entsprechen, was Ihr Frontend meldet.

- `handshake_messages()` — der einzelne gesendete `init`-Rahmen muss zur RADE-Initialisierung in `frontend/src/audio.js` passen. Ist dieser Handshake auseinandergelaufen, scheitert jede Zuhörerverbindung und der Test endet sofort mit Verbindungsfehlern.

Knickschwellen

Stellen Sie diese in `CONFIG` nach Bedarf ein — sie sind bewusst konservativ:

Schwelle	Standard	Stoppt, wenn
<code>TEMP_KNEE_C</code>	85.0	Die Package-Temperatur diese Obergrenze erreicht
<code>MIN_AVAIL_MB</code>	800	Der verfügbare Arbeitsspeicher darunter fällt
<code>SWAP_GROWTH_MB</code>	50	Der Swap über den beim Start genommenen Ausgangswert wächst
<code>FAIL_LIMIT</code>	3	So viele Zuhörer in einer Stufe keine Verbindung aufbauen
<code>MAX_LISTENERS</code>	60	Harter Stopp, auch wenn nie ein Knick erreicht wird

Die Form des Hochfahrens steuern `STEP` (je Stufe hinzugefügte Zuhörer, Standard 2), `SETTLE_S` (25 s Beruhigung vor der Messung — dadurch kann sich Wärme aufbauen) und `SAMPLE_S` (5 s CPU-Mittelungsfenster). Ein vollständiger Lauf bis zur Standardgrenze von 60 Zuhörern dauert daher rund 15 Minuten.

Die Ergebnisse lesen

Jede Stufe gibt eine Zeile aus und hängt eine Zeile an `rade_loadtest.csv` an:

Spalte	Bedeutung
<code>listeners</code>	In dieser Stufe verbundene synthetische Zuhörer
<code>connect_fails</code>	Zuhörer, die keine Verbindung aufbauen oder halten konnten
<code>pkg_c</code>	Package-Temperatur, Median aus 5 Messungen (spitzenresistent)
<code>cpu_pct</code>	CPU-Prozent des Gesamtsystems , nicht pro Zuhörer
<code>avail_mb</code>	Verfügbarer Arbeitsspeicher
<code>swap_mb</code>	Genutzter Swap
<code>dec_rss_mb</code>	Summierte RSS aller Decoder-Prozesse
<code>dec_procs</code>	Anzahl der Decoder-Prozesse — erwarten Sie 2 je Zuhörer (<code>rade_rxe.py</code> + <code>lpcnet_demo</code>)

Ein gemessener Lauf

Auf dem oben genannten i5-12450H endete ein vollständiger Lauf so:

```
KNEE at n=32: temp 91.0°C ≥ 85.0
Last stable concurrency: 30 RADE listeners
```

Die Wärme war die bindende Einschränkung, und zwar deutlich. Am Knick standen noch 7004 MB Arbeitsspeicher zur Verfügung — 6,2 GB oberhalb der `MIN_AVAIL_MB`-Grenze — und in jeder Stufe gab es null Verbindungsfehler.

Zwei Spalten lohnen genaues Lesen:

`dec_rss_mb` überschätzt die tatsächlichen Speicherkosten. Sie wächst sehr gleichmäßig um 295 MB je Zuhörer (292, 294, 294 ... 295 über alle 16 Stufen), doch der *verfügbare* Arbeitsspeicher sinkt nur um etwa **202 MB je Zuhörer**. Die beiden Prozesse hinter jedem Zuhörer teilen sich Bibliotheksseiten, und

die RSS zählt diese Seiten je Prozess einmal. Extrapoliert man die Steigung von `avail_mb`, käme der Speicherknick bei etwa **63 Zuhörern** — jenseits des harten Stopps bei 60 Zuhörern, auf dieser Maschine kann er also nie auslösen. Dimensionieren Sie den Speicher nach `avail_mb`, nicht nach `dec_rss_mb`.

cpu_pct ist flach — und dann nicht mehr. Sie liegt bis 24 Zuhörer bei ~5,3 % der gesamten Maschine und wird dann überlinear:

Zuhörer	24	26	28	30	32
cpu_pct	6,7 %	18,4 %	30,1 %	49,6 %	65,2 %

Das ist ein 10-facher Anstieg der CPU-Last, während die Zuhörerzahl um ein Drittel wächst. Der Wendepunkt reproduziert sich über Läufe hinweg an derselben Stelle, betrachten Sie ihn also als Eigenschaft der Maschine und nicht als Rauschen — die CPU-Kosten je Zuhörer liegen unter dem Wendepunkt bei rund 3,4 % eines Kerns und darüber bei etwa 20 %. Extrapolieren Sie keine CPU-Zahl je Nutzer, die bei geringer Zuhörerzahl gemessen wurde.

Der Test hat keinen CPU-Knick. Die vier Grenzwerte sind Temperatur, verfügbarer Arbeitsspeicher, Swap-Wachstum und Verbindungsfehler — die CPU wird erfasst, beendet den Lauf aber nie. Im obigen Lauf erreichte die CPU 65 % und wäre weiter gestiegen, hätte nicht zuerst die Temperatur ausgelöst. Wenn Ihnen eine CPU-Obergrenze wichtig ist, beobachten Sie die Spalte selbst oder ergänzen Sie eine Schwelle.

Was das belegt und was nicht. Jeder synthetische Zuhörer streamt zufälliges Rauschen geringer Amplitude, kein echtes RADE-Signal. Das genügt, damit die gesamte Dekodierkette läuft und Ressourcen verbraucht — die Ressourcenzahlen sind also aussagekräftig. Über die Dekodierqualität oder die Tonkontinuität unter Last sagt der Test jedoch nichts. Bestätigen Sie das mit echten Zuhörern an einem echten Signal.

Wenn Kerne vor dem Knick sättigen

RADE bindet je Instanz einen Thread fest, sodass einzelne Kerne ~90 °C erreichen können, während die Gesamt-CPU noch untätig wirkt. `rade_helper.py` verteilt die Ketten reihum über die Kerne, um dem entgegenzuwirken. Zeigt ein Lauf frühe thermische Knicke, erweitern Sie die Kernzuteilung je Client:

```
RADE_CORES_PER_CLIENT=3 ./rade.sh restart
```

Keine Dateiänderung nötig — das ist eine Umgebungsvariable (siehe [Umgebungsvariablen](#)); `RADE_PIN_CORES=0` deaktiviert das Pinning vollständig.

RADE v1 aktualisieren

Die Integration in PhantomSDR-Plus (`rade_helper.py`, Frontend-Patches) ist nur eine Brücke — die gesamte RADE-Dekodierlogik liegt im Repository `radac`. Aktualisierungen sind daher fast immer ein einfaches `git pull` plus Neubau, ohne Änderungen an PhantomSDR-Plus selbst.

Standardaktualisierung (neuer Code, gleiches Modell)

```
# 1. Pull latest radae code
cd ~/radae
git pull

# 2. Rebuild lpcnet_demo (in case C code changed)
cd build
cmake ..
make -j$(nproc)

# 3. Restart the sidecar – no server restart needed
cd ~/PhantomSDR-Plus
./rade.sh restart
```

Das ist alles. Kein Frontend-Neubau, kein Serverneustart, keine Dateiänderungen.

Nur neue Modellgewichte

Wird ein neuer Checkpoint veröffentlicht (z. B. model20) ohne Codeänderungen, richten Sie den Sidecar über die Umgebungsvariable auf die neuen Gewichte aus — ohne Datei zu bearbeiten:

```
# One-off: start with new model
RADE_MODEL=~/.radae/model20/checkpoints/checkpoint_epoch_100.pth ./rade.sh start

# Or permanently – add to your shell profile (~/.bashrc):
export RADE_MODEL=~/.radae/model20/checkpoints/checkpoint_epoch_100.pth
```

Was welche Änderung erfordert

Was sich in radae geändert hat	Erforderliche Maßnahme
Neue Modellgewichte (neue .pth-Datei)	RADE_MODEL setzen, ./rade.sh restart
Codeänderung in radae_rxe.py	git pull, ./rade.sh restart
C-Code von lpcnet_demo geändert	git pull, neu bauen, ./rade.sh restart
radae_rxe.py umbenannt oder verschoben	Pfad RADAЕ_RX in rade_helper.py anpassen (eine Zeile)
Argument --model_name umbenannt	radae_cmd in rade_helper.py anpassen (eine Zeile)
Neue Ausgabeabtastrate (≠ 16000 Hz)	SPS_OUT in rade_helper.py + createBuffer() in audio.js anpassen
RADE v2 nutzt eine andere Binärdatei	RADAЕ_RX in rade_helper.py anpassen (eine Zeile)

Prüfen, ob die Aktualisierung gegriffen hat

Bestätigen Sie nach dem Neustart, dass der neue Code läuft:

```
# Check sidecar picked up new radae_rxe.py
./rade.sh status

# Tail log to see startup lines
tail -20 ~/PhantomSDR-Plus/rade.log
```

Das Protokoll sollte den erwarteten Modellpfad zeigen:

```
[RADE] model : /home/sv1btl/radae/model19_check3/checkpoints/checkpoint_epoch_100.pth
```

Auf dem Laufenden bleiben

Abonnieren Sie die Releases-Seite des radae-Repositories, um über neue Modelle und Codeaktualisierungen benachrichtigt zu werden:

```
https://github.com/drowe67/radae/releases
```

Im FreeDV-Blog finden Sie Ankündigungen zu neuen RADE-Wellenformen und -Modellen:

```
https://freedv.org/blog/
```

Entwickelt und getestet auf PhantomSDR-Plus (Fork sv1btl/PhantomSDR-Plus) mit RX-888 MK2 unter Ubuntu 24.04. Der C++-Server wird nicht verändert.

RADE wird von David Rowe VK5DGR und dem FreeDV-Team entwickelt. Siehe freedv.org/radio-autoencoder.

Vollständige Deinstallation und Neuinstallation

Hier der komplette Rückbau, bevor `install_rade.sh` erneut ausgeführt wird:

1. Den Sidecar stoppen `cd ~/PhantomSDR-Plus && ./rade.sh stop`
2. Das radae-Repository und den Build entfernen `rm -rf ~/radae`
3. torch entfernen (im Benutzerverzeichnis installiert) `pip3 uninstall -y torch` `rm -rf ~/.local/lib/python3.11/site-packages/torch*`
4. Python-Pakete aus apt entfernen (optional — überspringen, wenn andere sie nutzen) `sudo apt-get remove -y python3-numpy python3-scipy python3-matplotlib python3-websockets` `sudo apt-get autoremove -y`
5. Prüfen, dass alles weg ist `python3 -c "import torch" 2>&1` # should say `ModuleNotFoundError` `ls ~/radae 2>&1` # should say `No such file or directory`
6. Neuinstallation `chmod +x ~/PhantomSDR-Plus/install_rade.sh` `~/PhantomSDR-Plus/install_rade.sh`

KAPITEL 10

RADE Allgemeine Installation (Linux)

RADE_General_INSTALL_MANUAL_LINUX.md

RADE v1 — Anleitung zur manuellen Installation

Alle Linux-Systeme — Ubuntu / Debian / Fedora / Arch / Raspberry Pi OS · PhantomSDR-Plus

Voraussetzung: Die gepatchten Dateien (`rade_helper.py`, `rade.sh`, `audio.js`, `App.svelte`) liegen bereits im Verzeichnisbaum von PhantomSDR-Plus. Diese Anleitung baut alles Übrige darum herum auf.

WICHTIG

Für eine normale Installation brauchen Sie diese Anleitung nicht. `./install_rade.sh` im PhantomSDR-Plus-Ordner erledigt jeden Schritt unten für Sie — Systempakete, Python-Module, den `radac`-Build, die Prüfung der Modellgewichte und den Start des Sidecars — und läuft auf `apt`-, `pacman`-, `dnf`- und `zypper`-Systemen gleichermaßen. `./install.sh` und die vier Distributionsskripte bieten an, es im Rahmen der normalen Installation auszuführen. Folgen Sie dieser Anleitung nur für eine manuelle Einrichtung, für ein System, das das Skript nicht abdeckt, oder um einen Schritt von Hand zu reparieren.

Raspberry-Pi-Nutzer: Dies ist auch Ihre Anleitung. Raspberry Pi OS ist Debian, folgen Sie also durchgehend den Debian-Blöcken und lesen Sie die Hinweise **Raspberry Pi / Bookworm**, wo sie auftauchen — sie behandeln die beiden Punkte, die sich auf einem Pi unterscheiden: PEP 668 und das CPU-only-torch-Wheel.

Schritt 1 — Systempakete

Wählen Sie den Block, der zu Ihrer Distribution passt.

Ubuntu / Debian / Raspberry Pi OS

```
sudo apt update
sudo apt install -y \
    build-essential cmake git \
    python3 python3-pip \
    nodejs npm \
    alsa-utils
```

Fedora / RHEL / Rocky

```
sudo dnf install -y \
    gcc gcc-c++ cmake git \
    python3 python3-pip \
    nodejs npm \
    alsa-utils
```

Arch / Manjaro

```
sudo pacman -Sy --needed \
    base-devel cmake git \
    python python-pip \
    nodejs npm \
    alsa-utils
```

Node.js-Version prüfen (alle Distributionen)

Der Frontend-Build von RADE benötigt Node.js 22 oder neuer:

```
node --version
```

Liegt die Version unter 22.x, installieren Sie Node.js über `nvm` (jede Distribution, ohne root):

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.4/install.sh | bash
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"
nvm install 22 && nvm use 22 && nvm alias default 22
```

Nicht über NodeSource. `deb.nodesource.com` und `rpm.nodesource.com` antworten inzwischen auf jedem Repository-Pfad mit HTTP 403. Die alte Zeile `curl -fsSL https://deb.nodesource.com/setup_NN.x | sudo -E bash -` funktioniert daher nicht mehr — und unter Debian/Ubuntu hinterlässt sie eine apt-Quelle, die jedes spätere `apt update` scheitern lässt. Aus demselben Grund verwenden die Installationsskripte `nvm`.

Schritt 2 — Python-Pakete

Ubuntu 24.04 und älter / Fedora / Arch

Auf diesen Systemen funktioniert ein einfaches `pip3 install`:

```
pip3 install websockets matplotlib numpy scipy

# torch – CPU-only build (~150–250 MB, avoids the ~3 GB CUDA wheel)
# Use this if the server has no GPU, which is the typical case for a WebSDR
pip3 install torch --index-url https://download.pytorch.org/whl/cpu
```

Verfügt Ihr Server **doch** über eine NVIDIA-GPU mit installiertem CUDA, können Sie `--index-url` weglassen, um den vollständigen CUDA-Build zu erhalten. Für RADE bringt das keinen Leistungsvorteil — `radar_xe.py` nutzt PyTorch nur für Matrixoperationen auf der CPU.

Ubuntu 23.04+ / Debian Bookworm+ / Raspberry Pi OS (PEP-668-Systeme)

Diese Distributionen blockieren ein bloßes `pip3 install`. Ergänzen Sie die Option:

```
# All packages except torch
pip3 install --break-system-packages websockets matplotlib numpy scipy

# torch – CPU-only build
pip3 install --break-system-packages torch \
    --index-url https://download.pytorch.org/whl/cpu
```

Raspberry Pi / Bookworm — zwei Regeln, die beißen:

1. Jedes `pip3 install` erfordert `--break-system-packages` (Durchsetzung von PEP 668). Ohne diese Option wird die Installation vollständig blockiert.
2. Ein einfaches `pip3 install torch` lädt das CUDA-Wheel (~3 GB) herunter. Auf einem Pi gibt es kein CUDA — nutzen Sie den CPU-Index, um das schlanke ARM64-Wheel (~150 MB) zu erhalten.

Alternativ nutzen Sie eine virtuelle Umgebung und vermeiden die Option ganz:

```
python3 -m venv ~/rade-venv
source ~/rade-venv/bin/activate
pip install websockets matplotlib numpy scipy torch \
  --index-url https://download.pytorch.org/whl/cpu
```

Bei Verwendung eines venv stellen Sie allen folgenden `python3`-Aufrufen dieser Anleitung `source ~/rade-venv/bin/activate` voran oder verwenden den vollen Pfad `~/rade-venv/bin/python3`.

Überprüfen

```
python3 -c "import websockets, matplotlib, torch, numpy, scipy; print('All OK')"
```

Erwartete Ausgabe:

```
All OK
```

Schritt 3 — Das radae-Repository klonen und bauen

Der RADE-Decoder liegt in einem anderen Repository als codec2. `freedv_rx` aus codec2 unterstützt RADE v1 **nicht**.

```
# Remove any previous incomplete clone
rm -rf ~/radeae

# Clone
git clone https://github.com/drowe67/radae.git ~/radeae
cd ~/radeae

# Build
mkdir build && cd build
cmake ..
make -j$(nproc)
```

Prüfen, ob lpcnet_demo gebaut wurde

```
ls -la ~/radeae/build/src/lpcnet_demo
```

Erwartet: Die Binärdatei ist vorhanden und ausführbar.

Prüfen, ob die Modellgewichte vorhanden sind

```
ls ~/radeae/model19_check3/checkpoints/checkpoint_epoch_100.pth
```

Erwartet: Die `.pth`-Datei wird aufgelistet. Die Gewichte liegen dem Repository bei — ein separater Download ist nicht nötig.

Schritt 4 — Die Dekodierkette überprüfen

Dieser Schritt bestätigt, dass die vollständige Offline-Kette funktioniert, bevor sie mit dem Browser verbunden wird. Ausführen aus `~/rade`:

```
cd ~/rade
```

4a — Ein RADE-kodiertes Testsignal erzeugen

```
./inference.sh model19_check3/checkpoints/checkpoint_epoch_100.pth \
wav/brian_g8sez.wav /dev/null \
--rate_Fs --pilots --pilot_eq --eq_ls --cp 0.004 \
--write_rx rx.f32 --auxdata
```

Warten Sie, bis der Vorgang beendet ist. Die letzten ausgegebenen Zeilen sollten lauten:

```
loss: 0.741 Auxdata BER: 0.012
```

4b — Dekodieren und abspielen

```
cat rx.f32 \
| python3 rade_rxe.py \
--model_name model19_check3/checkpoints/checkpoint_epoch_100.pth \
| ./build/src/lpcnet_demo -fargan-synthesis - - \
| aplay -f S16_LE -r 16000
```

Sie sollten eine Stimme hören. Die Ausgabe zeigt das Einrasten der Synchronisation:

```
1 state: search    ...
5 state: sync      ... SNRdB: 4.03 uw_err: 0
Playing raw data 'stdin' : Signed 16 bit Little Endian, Rate 16000 Hz, Mono
```

underrun!!!-Meldungen sind bei diesem Test zu erwarten und harmlos. Sie entstehen, weil `rade_rxe.py` langsamer verarbeitet als die Datei-Ein-/ Ausgabe. Beim Live-Empfang treten sie nicht auf — der Browser liefert den Ton in Echtzeit.

4c — Prüfen, ob der Sidecar korrekt startet

```
python3 ~/PhantomSDR-Plus/rade_helper.py
```

Erwartete Ausgabe (keine WARNING-Zeilen):

```
[RADE] helper starting on ws://0.0.0.0:8074
[RADE] rade_rx.py      : /home/<user>/rade/rade_rxe.py
[RADE] model          : /home/<user>/rade/model19_check3/checkpoints/checkpoint_epoch_100.pth
[RADE] lpcnet_demo     : /home/<user>/rade/build/src/lpcnet_demo
[RADE] auxdata         : ON (default)
[RADE] torch threads  : 1 per instance (RADE_TORCH_THREADS to override)
[RADE] architecture   : per-connection (each user tunes independently)
[RADE] listening      : waiting for PhantomSDR-Plus clients
```

Drücken Sie **Strg+C** zum Beenden.

<user> in den Pfaden oben ist das Konto, mit dem Sie angemeldet sind — auf einem Standard-Image von Raspberry Pi OS lauten sie damit `/home/pi/radae/...`.

Schritt 5 — Das PhantomSDR-Plus-Frontend bauen

```
cd ~/PhantomSDR-Plus/frontend

# Only if node_modules is missing:
npm install

cd ~/PhantomSDR-Plus
./recompile.sh
```

Achten Sie auf Parserfehler von Vite/acorn. Die gepatchten Dateien vermeiden bewusst `?.`, `??` und leere `catch {}`, um die acorn-Einschränkung einzuhalten.

Schritt 6 — Steuerung des Sidecars

Normalerweise brauchen Sie kein eigenes Steuerskript für RADE. Die Startskripte (`start-rx888mk2.sh`, `start-airspyhf.sh`, `start-rtl.sh`, `start-rspla.sh`, `start-fobos-hf.sh`, `start-fobos.sh`, `start-hackrf.sh`) starten das Sidecar selbst, sobald der Server läuft, starten es neu, wenn es sich beendet, und melden seinen Zustand; `stop-websdr.sh` stoppt es zusammen mit dem Server. Gehen Sie direkt zu Schritt 7.

`rade.sh` wird weiterhin für den eigenständigen Betrieb mitgeliefert — das Sidecar ohne den PhantomSDR-Plus-Launcher zu betreiben oder es allein zu testen. Seine Befehle, sein Protokoll und die zugehörigen Watchdog-Fallstricke stehen unter Steuerung des Sidecars.

Schritt 7 — Port 8074 öffnen

Der Sidecar lauscht auf TCP-Port **8074**. Der Browser verbindet sich direkt mit diesem Port. Sie müssen ihn selbst öffnen.

ufw (Ubuntu / Debian / Raspberry Pi OS)

```
sudo ufw allow 8074/tcp && sudo ufw reload
```

firewalld (Fedora / RHEL / Rocky)

```
sudo firewall-cmd --add-port=8074/tcp --permanent
sudo firewall-cmd --reload
```

iptables (jede Distribution, dauerhaft)

```
sudo iptables -I INPUT -p tcp --dport 8074 -j ACCEPT

# Ubuntu / Debian / Raspberry Pi OS – persist across reboots:
sudo apt install iptables-persistent
sudo netfilter-persistent save

# Fedora / RHEL – persist:
sudo service iptables save
```

Router

Legen Sie eine NAT-/Portweiterleitungsregel an: **TCP 8074 → LAN-IP des Servers : 8074**

Test von außen

Verwenden Sie **https://portchecker.co** und prüfen Sie Port 8074 gegen Ihren öffentlichen Hostnamen. Testen Sie **nicht** mit `curl` vom Server selbst — NAT- Hairpin liefert falsche „Connection refused“-Ergebnisse, selbst wenn der Port offen ist.

Alternative — Nginx-Proxy (falls Port 8074 vom Provider gesperrt ist)

Fügen Sie innerhalb Ihres bestehenden `server {}`-Blocks ein:

```
location /rade {
    proxy_pass          http://127.0.0.1:8074;
    proxy_http_version 1.1;
    proxy_set_header    Upgrade      $http_upgrade;
    proxy_set_header    Connection   "upgrade";
    proxy_set_header    Host         $host;
    proxy_read_timeout  3600s;
}
```

```
sudo nginx -t && sudo nginx -s reload
```

Bearbeiten Sie danach `audio.js` innerhalb von `setRADEDecoding()`:

```
// Change:
var helperUri = uri || ('ws://' + window.location.hostname + ':8074');
// To:
var helperUri = uri || ('ws://' + window.location.host + '/rade');
```

Bauen Sie das Frontend nach dieser Änderung neu (`./recompile.sh`).

Schritt 8 — Den Server starten

Starten Sie PhantomSDR-Plus wie gewohnt. RADE wird **nicht** automatisch gestartet:

```
cd ~/PhantomSDR-Plus
./start-rx888mk2.sh      # or start-airspyhf.sh,
                        # start-rtl.sh, start-rspla.sh,
                        # start-fobos-hf.sh, start-fobos.sh,
                        # start-hackrf.sh
```

Verwenden Sie den Starter, der zu Ihrem Empfänger passt. Jeder bringt einen eigenen Watchdog und ein eigenes Protokoll mit; `./stop-websdr.sh` beendet den jeweils laufenden.

Schritt 9 — RADE starten

```
cd ~/PhantomSDR-Plus
./rade.sh start
./rade.sh status
tail -f rade.log
```

Erwartetes Protokoll:

```
[RADE] sidecar starting at ...
[RADE] helper starting on ws://0.0.0.0:8074
[RADE] rade_rx.py      : /home/<user>/rade/rade_rxe.py
[RADE] model          : /home/<user>/rade/model19_check3/checkpoints/checkpoint_epoch_100.pth
[RADE] lpcnet_demo     : /home/<user>/rade/build/src/lpcnet_demo
[RADE] auxdata        : ON (default)
[RADE] torch threads  : 1 per instance
[RADE] architecture   : per-connection (each user tunes independently)
[RADE] listening      : waiting for PhantomSDR-Plus clients
```

Schritt 10 — RADE im Browser nutzen

1. Öffnen Sie Ihre PhantomSDR-Plus-Weboberfläche
2. Aktive Stationen finden Sie unter **<https://qso.freedv.org>**
3. Stimmen Sie auf die Anzeigefrequenz der Station ab
4. Wählen Sie unter **Decoder Options**:
5. **RADE v1 — RADEL (LSB)** für 40 m / 80 m / 160 m (≤ 10 MHz)
6. **RADE v1 — RADEU (USB)** für 20 m / 17 m / 15 m / 12 m / 10 m (> 10 MHz)
7. Klicken Sie auf **Decoder: ON**

Die Schritte 4 und 5 lassen sich durch einen einzigen Druck ersetzen: Die Tasten **RADEL** / **RADEU** stehen neben der Überschrift **Modes selector** sowie in den Popup-Fenstern **Modes** und **Bands**. Ein Druck wählt den Decoder aus, schaltet ihn ON und holt das RADE-Panel in den sichtbaren Bereich; ein erneuter Druck schaltet RADE ab. Siehe das [RADE-Handbuch](#).

Anzeige	Bedeutung
Rot — „Connecting to sidecar...“	Port 8074 nicht erreichbar oder Sidecar läuft nicht
Gelb — „Searching for signal...“	Sidecar verbunden, noch kein RADE-Rahmen erkannt (etwa 1,5 s einplanen)
Grün — „Synced · SNR x.x dB“	Dekodierung läuft — Sprache ist zu hören

Hinweis zur CPU-Last

Jeder RADE-Nutzer belegt etwa 8–10 % eines Kerns (die PyTorch-Threadzahl ist vom Sidecar auf 1 begrenzt). Falls Sie Tonaussetzer hören, erhöhen Sie auf 2 Threads:

```
RADE_TORCH_THREADS=2 ./rade.sh restart
```

Gleichzeitige Nutzer	Ungefähre CPU-Last
1	~9 %
5	~45 %
10	~90 %
20	~180 %

Diese Werte wurden auf x86_64 gemessen. Ein Raspberry Pi dekodiert RADE problemlos, doch die Kosten pro Nutzer sind höher und die Tabelle oben lässt sich nicht übertragen — messen Sie Ihr eigenes Board mit `top`, während ein Nutzer synchronisiert ist, bevor Sie eine Nutzerzahl angeben.

RADE künftig aktualisieren

```
cd ~/radae
git pull
cmake -S . -B build && make -C build -j$(nproc)

cd ~/PhantomSDR-Plus
./rade.sh restart
```

Weder ein Neubau des Frontends noch ein Serverneustart ist nötig, sofern sich nicht `rade_helper.py` selbst geändert hat.

Getestet auf Ubuntu 24.04 (x86_64) und Raspberry Pi 4 / Raspberry Pi OS Bookworm (ARM64, Python 3.11). PhantomSDR-Plus-Fork: [sv1btl/PhantomSDR-Plus](https://github.com/sv1btl/PhantomSDR-Plus). RADE entwickelt von David Rowe VK5DGR und dem FreeDV-Team — <https://freedv.org>

KAPITEL 11

Varianten bearbeiten

EDITING_VARIANTS.md

PhantomSDR-Plus — Bearbeiten der Frontend-Varianten

(nachdem die vier Dateien `App__*_smeter_.svelte` zu einer einzigen `App.svelte` zusammengeführt wurden)

1. Was die Varianten sind

Zwei unabhängige Entscheidungen, die vier Kombinationen ergeben:

```
smeter = "analog"    das Instrument mit beweglichem Zeiger
      = "digital"    das segmentierte Balkeninstrument

layout = "v1"        Betriebsartenwahl oben, Bandwahl unten
      = "v2"        Bandwahl oben, Betriebsartenwahl unten
                    (verschiebt auch die Feinabstimmungs-Tastenreihe auf dem Handy)
```

Alle vier stecken in **einem** Build, ausgeliefert unter `/`. Der Wähler oben rechts schaltet zwischen ihnen um, während die Seite läuft: Er setzt die Props `smeter/layout` von `App.svelte`, es wird also nichts neu geladen, und Audio, Wasserfall und Decoder laufen weiter. Die Wahl wird pro Browser in `localStorage` unter `phantom.variant` gespeichert und hat beim nächsten Besuch Vorrang vor dem Build-Standard.

Die in `recompile.sh` gewählte Variante ist damit die **Start**variante — das, was ein Erstbesucher sieht, bevor er das Menü anfasst.

URL	was es ist
<code>/</code>	die Desktop-Seite, alle vier Varianten, zur Laufzeit umschaltbar
<code>/mobile/</code>	eigene Seite, gebaut von <code>build-mobile.sh</code> — keine Variante

Bis zum 10.08.2026 war jede Variante ein eigener Build unter `/analog/`, `/digital/`, `/v2-analog/` und `/v2-digital/`, und der Wähler navigierte zwischen ihnen — deshalb lud das Umschalten früher die Seite neu und riss den Ton ab. Diese Builds gibt es nicht mehr; unter jedem der vier Pfade liegt nur noch eine kleine `index.html`, die auf `/` weiterleitet — geschrieben von `frontend/make-redirect-stubs.sh` bei jedem Build, damit alte Lesezeichen weiter funktionieren.

Die Variante wird in **keine** Quelldatei geschrieben. `src/main.js` ist ein fester Einstiegspunkt, der der Komponente zwei Build-Zeit-Defines übergibt:

```
const app = new App({
  target: document.getElementById('app'),
  props: { smeter: __PHANTOM_SMETER__, layout: __PHANTOM_LAYOUT__ }
})
```

`vite.config.js` füllt diese beim Bauen aus und nimmt dabei das erste der folgenden, das gesetzt ist:

1. **PHANTOM_SMETER** / **PHANTOM_LAYOUT** — Umgebungsvariablen. Seit dem Wegfall der Varianten-Builds setzt sie niemand mehr; für einen einmaligen Build mit einer anderen Startvariante funktionieren sie weiterhin.
2. **frontend/variant.json** — der Standard der Website, geschrieben von `recompile.sh`. Der Build für die Wurzel (/) übergibt keine Variante und landet daher hier.
3. **analog** / **v1** — der Rückfallwert.

Diese beiden Werte sind nur der **Startwert**. `App.svelte` überschreibt sie beim Initialisieren aus `localStorage`, wenn der Besucher bereits eine Variante gewählt hat, und der ⚙-Wähler setzt sie bei jedem Umschalten erneut.

In `App.svelte` werden die Props in den Zeilen 20-21 in zwei Flags umgewandelt:

```
$: isAnalog = smeter !== "digital";
$: isV2      = layout === "v2";
```

Verwenden Sie diese beiden Flags — niemals `smeter/layout` direkt — damit sich die ganze Datei einheitlich liest.

2. Welche Datei was macht

`frontend/src/main.js` — 12 Zeilen

Der Einstiegspunkt. Nichts außer der Instanziierung von `App` und den zwei `__PHANTOM_*__`-Defines, die sie als Props weiterreicht. Sie ist **konstant** — weder `recompile.sh` noch die Build-Skripte schreiben sie noch um, und genau das erlaubt es, die Varianten gleichzeitig zu bauen.

Legen Sie hier keinen Anwendungscode ab. Es ist die Einstiegsdatei, kein Ort für Logik.

`frontend/src/App.svelte` — ca. 17130 Zeilen

Die gesamte Anwendung: Wasserfall, Abstimmung, Dekoder, Lesezeichen, Bedienfelder, Desktop-Layout UND das responsive Handy-Layout. Nahezu jede Änderung, die Sie je vornehmen werden, gehört hierher.

`frontend/src/lib/SMeterAnalog.svelte` — 824 Zeilen

Das Zeigerinstrument, vollständig: Zeichnen des Zifferblatts, Zeiger, die Umschaltung hell/dunkel samt zugehörigem `localStorage`-Schlüssel und die Zeigerglättung.

- Eingangs-Prop: `dbm` — die KALIBRIERTE Leistung in dBm.
- Nimmt außerdem: `mobile` — kleinere Canvas-Attribute für das Handy-Layout.
- Stellschraube: `SMOOTH_TIME_MS` in Zeile 55. Kleiner = schnellerer Zeiger, größer = ruhiger. Derzeit 16 (ein Frame bei 60 fps, praktisch die Untergrenze).

Die drei Skalenbilder — und `smeter_theme.sh`

Das analoge Instrument hat drei Hintergründe, alle von demselben Code gezeichnet:

<code>dark</code>	das ursprüngliche dunkle, gebürstete Metall
<code>amber</code>	ein helles, blassgraues Zifferblatt
<code>vintage</code>	ein warmes, gealtertes Bernstein-Zifferblatt

Nicht mit der Variante `smeter = analog / digital` oben verwechseln. Die Variante wählt, *welche Instrumentenkomponente* angezeigt wird; das Skalenbild wählt, *wie das analoge gezeichnet* wird. Sie liegen unter verschiedenen `localStorage`-Schlüsseln (`phantom.variant` bzw. `smeterTheme`) und werden an verschiedenen Stellen gesetzt.

Die Besucherseite. Das Canvas des Instruments ist eine Schaltfläche (`role="button"`, auch Enter/Leertaste über die Tastatur, mit dem Hinweis „click to switch style“ darunter). Jeder Klick schaltet `dark` → `amber` → `vintage` weiter und speichert die Wahl in `localStorage.smeterTheme`. Sie wird bei jedem Seitenaufbau zurückgelesen, überlebt also Neuladen und Browser-Neustarts unbegrenzt. Bewusst **kein** Cookie: ein Cookie würde bei jeder Wasserfall- und Audio-Anfrage an den Server geschickt, für einen Wert, den der Server nie liest.

Die Sysop-Seite. Zwei Dinge in `SMeterAnalog.svelte` bestimmen, was alle anderen bekommen:

Zeile	Wirkung
<code>let smeterTheme = 'dark';</code>	Das Skalenbild, das ein Browser ohne gespeicherte Wahl bekommt.
<code>const SMETER_PREF_VERSION = 2;</code>	Ein Zähler. Liegt die gespeicherte Version eines Browsers darunter, vergisst er sein gespeichertes Skalenbild einmalig und übernimmt die obige Vorgabe. Erhöhen Sie ihn, um Besucher umzustellen, die bereits gewählt haben.

Nur die Vorgabe zu ändern erreicht also ausschließlich ganz neue Besucher. Beide Änderungen zusammen erreichen alle — genau dafür gibt es `smeter_theme.sh` im Wurzelverzeichnis:

```
./smeter_theme.sh          # aktuelle Vorgabe anzeigen, dann aus einem Menü wählen
./smeter_theme.sh vintage  # direkt setzen, danach den Neubau anbieten
./smeter_theme.sh dark --build # frontend/build-all.sh ohne Rückfrage ausführen
./smeter_theme.sh amber --no-reset # nur neue Besucher – bestehende in Ruhe lassen
```

Es meldet die aktuelle Vorgabe, setzt die neue, erhöht den Reset-Zähler (immer aufwärts — **niemals verringern**, eine kleinere Zahl verhindert das Zurücksetzen einfach) und bietet dann dieselbe Frontend-Bauauswahl wie `recompile.sh`. Es fasst weder das Backend an noch schreibt es `variant.json` neu.

Eine Änderung erreicht einen verbundenen Hörer erst beim Neuladen seiner Seite — die Vorgabe ist ins JS-Bundle kompiliert und das Zurücksetzen läuft beim Laden. Es gibt keine Live-Übertragung. Ein normales Neuladen genügt, da vite die Bundle-Dateinamen hasht.

Das einmalige Zurücksetzen des Zählers überschreibt alle, auch Besucher, die bewusst ihr eigenes Skalenbild angeklickt haben. Das ist beabsichtigt: nur so lässt sich die ganze Seite wieder auf ein Aussehen bringen. Mit `--no-reset` lassen Sie sie in Ruhe.

frontend/src/lib/SMeterDigital.svelte — 184 Zeilen

Das Balkeninstrument, vollständig.

- Eingangs-Props: `rawDb` — der ROHE Wert von `audio.getPowerDb()`
- `smeterOffset` — `audio.smeter_offset`
- `mobile` — kleinere Canvas-Attribute

HINWEIS: Dieses Instrument wird bewusst aus der ROHEN Leistung plus Offset gespeist, NICHT aus dem kalibrierten dBm-Wert des analogen Zifferblatts. Es hat schon immer seine eigene Abbildung gehabt. Ihm den kalibrierten Wert zu geben, würde auf jeder digitalen Station jede Anzeige verschieben. Diese beiden Eingänge nicht „vereinheitlichen“.

frontend/src/lib/BandSelector.svelte — 69 Zeilen

Das Raster der Bandtasten. Wird in `App.svelte` zweimal gerendert (einmal je Layout).

frontend/src/lib/ModesSelector.svelte — 75 Zeilen

Das Raster der Betriebsartentasten. Wird in `App.svelte` zweimal gerendert.

frontend/src/lib/StatusIndicators.svelte — 122 Zeilen

Die Statuslampen (Mute, Squelch, NR, NB, NS, AN, CTCSS). Nimmt ein `wide`-Prop: Das digitale Layout verwendet breitere Lampen als das analoge. Beide Tailwind-Klassen (`w-8` und `w-10`) stehen ausgeschrieben in der Komponente — Tailwinds Scanner sieht nur wörtliche Klassennamen, bauen Sie sie also niemals durch Zeichenkettenverkettung zusammen.

Alle übrigen Dateien in `frontend/src/lib/` (Spectrogram, PassbandTuner, FrequencyInput, VersionSelector, ...) stammen aus der Zeit vor dieser Arbeit und sind unverändert.

3. Eine Änderung, die nur einige Varianten betrifft

Packen Sie sie in das Flag. Im Markup:

```
{#if isAnalog}
  ...nur die analogen Stationen sehen das...
{:else}
  ...nur die digitalen Stationen sehen das...
{/if}

{#if isV2} ... {/if}      {#if !isV2} ... {/if}
```

In einem class-Attribut:

```
class="p-4 {isAnalog ? 'text-xs' : 'text-sm'} rounded-md"
```

Schreiben Sie beide Alternativen wie oben als vollständige Literale aus. Tailwind durchsucht den Quelltext nach vollständigen Klassennamen; `text-{size}` oder ein zusammengesetzter Name wird aus dem CSS entfernt, und die Formatierung verschwindet stillschweigend.

Im `<script>`-Block: Versuchen Sie, gar nicht zu verzweigen. Das Skript berechnet bewusst bei jedem Tick die Eingaben BEIDER Instrumente —

```
smeterDbm      = powerDb;           // analoges Zifferblatt
smeterRawDb    = audio.getPowerDb(); // digitales Instrument
smeterOffset   = audio.smeter_offset; // digitales Instrument
```

— sodass nur das Markup auswählen muss. Deshalb gibt es in einer Datei mit 17000 Zeilen nur 15 Variantenweichen. Halten Sie das so; Verzweigungen auf Skriptebene haben die vier alten Dateien auseinanderdriften lassen.

4. Wo die Variantenweichen sitzen

Fünfzehn Stellen in `App.svelte`. Zeilennummern verschieben sich beim Bearbeiten — der zuverlässige Weg, sie alle zu finden, ist:

```
cd frontend/src && grep -n 'isAnalog\|isV2' App.svelte
```

Zum Zeitpunkt der Erstellung:

Zeile	Flag	Was umgeschaltet wird
20		isAnalog deklariert
21		isV2 deklariert
8062	isV2	Desktop: Bandwahl zuerst (v2-Reihenfolge)
8090	isV2	Desktop: Schriftgröße der Bedienfeldüberschrift
8686	isV2	Desktop: Betriebsartenwahl zuerst (v1-Reihenfolge)
8717	isAnalog	Mindestbreite des Instrumentenfelds (der Zeiger ist schmaler)
8730	isAnalog	nur analog: Datums-/Zeitzeile über dem Instrument
8819	isAnalog	nur analog: Statuslampen + Trennlinie
8834	isAnalog	nur digital: Zeitzeile, breite Lampen und die Wahl zwischen <SMeterDigital> / <SMeterAnalog> selbst
8861	isAnalog	vertikale Abstände des Frequenzblocks
8864	isAnalog	oberer Rand des Feinabstimmungsblocks
12288	isV2	Handy: Feinabstimmungsreihe oben (v2-Reihenfolge)
12347	isAnalog	Handy: welches Instrument gerendert wird
12433	isV2	Handy: Feinabstimmungsreihe unten (v1-Reihenfolge)
13204	isAnalog	Schriftgröße der Noise-Gate-Taste

Alles Übrige in der Datei ist allen vier Varianten gemeinsam.

5. Weitere wissenswerte Stellschrauben

Ort	Konstante	Zweck
App.svelte Zeile 5034	<code>const visualGain = 1.1;</code>	Kalibrierung des analogen Instruments. Je 0,10 entspricht etwa 5 dBm in der Anzeige.
SMeterAnalog.svelte Zeile 55	<code>const SMOOTH_TIME_MS = 16;</code>	Zeitkonstante der Zeigerglättung in Millisekunden.
SMeterAnalog.svelte Zeile 64	<code>let smeterTheme = 'dark';</code>	Standard-Skalenbild des analogen Instruments für einen Browser ohne gespeicherte Wahl. Mit <code>./smeter_theme.sh</code> setzen, nicht von Hand.
SMeterAnalog.svelte Zeile 69	<code>const SMETER_PREF_VERSION = 2;</code>	Einmaliger Reset-Zähler für das gespeicherte Skalenbild. Erhöhen, um bestehende Besucher auf die Vorgabe zu holen; wird nur größer.
SMeterDigital.svelte Zeile 149	<code>const DIGITAL_BAR_TRIM = 0;</code>	Verschiebt den Balken um ganze Segmente.
SMeterDigital.svelte Zeile 31	<code>const numberOfDots = 35;</code>	Anzahl der Segmente des Balkens.

6. Nach dem Bearbeiten — neu bauen

Am einfachsten aus dem Wurzelverzeichnis des Projekts:

```
./recompile.sh
```

Menüpunkt **[2]** baut das Frontend neu und fragt, welche Variante ein Erstbesucher sehen soll. Diese Wahl wird in `frontend/variant.json` festgehalten — es wird keine Quelldatei kopiert oder umgeschrieben. (Früher wurde zusätzlich `VersionSelector.svelte` aus einem Heredoc neu erzeugt; das wurde entfernt, als der Wähler aufhörte zu navigieren, denn es hätte das Umschalten zur Laufzeit bei jedem Lauf zurückgesetzt.) Menüpunkt **[1]** ist nur das Backend, **[3]** beides.

Oder direkt, aus `frontend/`:

```
./build-all.sh      baut die Desktop-Seite UND /mobile/ &lt;-- normalerweise dieses
./build-default.sh   nur die Desktop-Seite
./build-mobile.sh    nur /mobile/
```

`build-default.sh` lässt vite `dist/` leeren, wodurch `dist/mobile/` bis zum nächsten `build-all.sh` oder `build-mobile.sh` verschwindet. Deshalb ist `build-all.sh` die normale Wahl.

`build-all.sh` besitzt eine eigene Kopie der Build-Logik (`build_version` SMETER LAYOUT NAME OUTDIR BASE LOG), statt `build-default.sh` aufzurufen. Wenn Sie ändern, wie die Seite gebaut wird, ändern Sie es an BEIDEN Stellen, sonst laufen die beiden Wege auseinander.

Ein paar wissenswerte Details:

- Es setzt `PHANTOM_KEEP_OUTDIR=1`, damit Vite das Ausgabeverzeichnis nicht leert, und räumt `dist/` stattdessen selbst auf — es löscht den Inhalt, behält aber `users.json`, das der laufende `spectrumserver` bei jedem Verbinden und Trennen eines Hörers neu schreibt.
- Es baut `/mobile` zuletzt, wenn die Desktop-Ausgabe vollständig ist.
- Seine Parallel-Build-Mechanik (`PHANTOM_BUILD_JOBS`, Standard 3) stammt aus der Zeit mit fünf Desktop-Builds. Bei einem einzigen steht nichts mehr in der Warteschlange.

Ausführbar-Bits: `recompile.sh` führt vor dem Start `chmod +x` auf ein Skript aus, und `build-all.sh` ruft bewusst `bash build-mobile.sh` statt `./build-mobile.sh` auf — ein fehlendes Bit bricht den Build also nicht mehr. Das Hochladen eines Skripts über die GitHub-Weboberfläche setzt es allerdings weiterhin auf 644 zurück; wenn Sie eines direkt starten:

```
chmod +x frontend/build-*.sh
```

7. Wie Sie Ihre Änderung prüfen, bevor Sie ihr vertrauen

Ein grüner vite build genügt **NICHT**. Der Build kann einen Bezeichner nicht sehen, der beim Verschieben von Code zwischen Dateien verlorengeht — das ist ein `ReferenceError` zur Laufzeit, und die Seite lädt schlicht nicht. Das ist bereits zweimal passiert.

1) Prüfung auf undefinierte Bezeichner (fängt genau diesen Fehler ab)

```
cd frontend
npx eslint --no-eslintrc \
  --parser svelte-eslint-parser \
  --rule '{"no-undef":"error"}' \
  --env browser,es2022 \
  src/App.svelte src/lib/SMeter*.svelte
```

Ein bekannter, harmloser Treffer ist zu erwarten:

```
'dBmCalOffset' is not defined
```

`typeof x === 'number'` auf einem nicht deklarierten Namen wirft nie einen Fehler, dieser Ausdruck ergibt also bereits 0. Alles andere ist ein echtes Problem.

2) Bauen, dann die SEITE LADEN UND ALLE VIER VARIANTEN DURCHSCHALTEN

Nicht nur die, die Sie geändert haben — eine Änderung unter `isAnalog/isV2` kann die anderen drei zerstören, während Ihre funktioniert. Öffnen Sie `/`, gehen Sie alle vier Einträge im ⚙️-Menü durch und öffnen Sie dann `/mobile/`.

Öffnen Sie auf jeder die Browserkonsole. Eine leere oder halb gezeichnete Seite mit einem `ReferenceError` in der Konsole ist die Signatur des oben beschriebenen Fehlers.

3) Lassen Sie `frontend/dist/` niemals in einem Teilzustand

Ein blankes `npx vite build` leert das Ausgabeverzeichnis und nimmt `dist/mobile/` sowie das laufende `dist/users.json` des Servers mit — die Mobilseite antwortet mit 404, bis der nächste Build erfolgreich ist. Bauen Sie immer über die Skripte: `build-all.sh` setzt `PHANTOM_KEEP_OUTDIR=1` und räumt selbst auf, wobei `dist/users.json` erhalten bleibt. Führen Sie im Zweifel einfach erneut `./build-all.sh` aus.

KAPITEL 12

Benutzerhandbuch

USER_GUIDE.md

PhantomSDR-Plus Benutzerhandbuch

Willkommen bei PhantomSDR-Plus! Dieses Handbuch hilft Ihnen, das Beste aus Ihrem WebSDR-Hörerlebnis herauszuholen.

Inhaltsverzeichnis

1. Einführung
 2. Erste Schritte
 3. Überblick über die Oberfläche
 4. Grundlegende Bedienung
 5. Erweiterte Funktionen
 6. Demodulationsarten
 7. Digitale Decoder
 8. Tastenkürzel
 9. Lesezeichen
 10. Nutzung auf Mobilgeräten
 11. Tipps und bewährte Praktiken
 12. Fehlerbehebung
 13. Häufig gestellte Fragen
-

Einführung

Was ist PhantomSDR-Plus?

PhantomSDR-Plus ist ein webbasiertes Software-Defined Radio (SDR), mit dem Sie Funksignale über das Internet hören können. Auf Ihrer Seite werden weder besondere Software noch Hardware benötigt – nur ein moderner Webbrowser!

Was können Sie hören?

Je nach Konfiguration des WebSDR können Sie Folgendes empfangen:

- **Amateurfunk:** Funkamateure weltweit
- **Rundfunkstationen:** AM-/UKW-Rundfunk, Kurzwellenrundfunk
- **Luftfahrt:** Flugsicherung, Flugfunk
- **Seefunk:** Schiff-Land-Verkehr, Seewetterdienst
- **Wettersatelliten:** NOAA, METEOR-M
- **Digitale Betriebsarten:** FT8, RTTY, PSK31 und mehr
- **Nutzfunkstationen:** Zeitzeichen, Militär, Behörden

Systemanforderungen

- **Browser:** Chrome/Edge (empfohlen), Firefox, Safari
 - **Verbindung:** Breitband-Internet (1 Mbit/s oder mehr empfohlen)
 - **Audio:** funktionierende Lautsprecher oder Kopfhörer
 - **Optional:** Maus mit Scrollrad zum leichteren Abstimmen
-

Erste Schritte

1. Auf das WebSDR zugreifen

Öffnen Sie Ihren Browser und rufen Sie die vom Betreiber angegebene WebSDR-Adresse auf.

Beispiel: `http://websdr.example.com:9002`

2. Erster Seitenaufbau

Beim Laden der Seite sehen Sie:

- eine farbige Wasserfallanzeige mit der Funkaktivität
- ein Bedienfeld mit Frequenzanzeige und Schaltflächen
- ein S-Meter für die Signalstärke
- eine Anzeige der Nutzerzahl

3. Mit dem Hören beginnen

1. **Klicken Sie auf ein Signal** in der Wasserfallanzeige
 2. **Der Ton startet automatisch**
 3. **Regeln Sie die Lautstärke** über die Lautstärkeregelung des Browsers oder den Regler auf dem Bildschirm
 4. **Stimmen Sie fein ab**, indem Sie genau auf das Signal klicken
-

Überblick über die Oberfläche

Hauptbestandteile

WebSDR <callsign>, located in <QTH> ⓘ [Analog S-Meter ▼] email · Keyboard Shortcuts · Users · Stats · Other servers 1 2 3 Frequency Search: MW SW HamDash PSK Reporter WSPRnet · Callsign Search [Open Additional Info]			
Spectrum + Waterfall band-plan strip along the bottom			
Audio & Buffer volume · SQ · zoom AGC Auto Fast Med Slow Compressor Equalizer	Frequency 7,120.00 band · VFO	S-Meter dBm dBµV SNR · NF	Waterfall Controls Min · Max · zoom colour-map strip Zoom In Out Max Min
Filters NR NB NS AN CTCSS Modes selector ... RADEL RADEU USB LSB CW AM QUAM FM Band selector	Fine Tuning (kHz) VFO A · Modes · Bands · IF Filters Wheel Tuning Steps Decoders FT8 FT4 FT2 JS8 CW WSPR FAX SSTV NAVTEX RTTY ...	Waterfall Spectrum Auto Adj. Height [profile ▼] [Bookmarks]	

Die drei Panels unter dem Wasserfall scrollen als eine Seite; in einem schmalen Fenster stapeln sie sich, statt nebeneinander zu stehen. **RADEL** und **RADEU** sitzen auf der Überschriftenzeile *Modes selector* selbst, rechts davon.

1. Wasserfallanzeige

Der Wasserfall ist eine visuelle Darstellung von Funksignalen:

- **Waagerechte Achse:** Frequenz
- **Senkrechte Achse:** Zeit (läuft nach unten)
- **Farben:** Signalstärke
- **Dunkelblau/schwarz:** kein Signal (Rauschteppich)
- **Grün/gelb:** schwache bis mittlere Signale
- **Orange/rot:** starke Signale
- **Weiß:** sehr starke Signale

2. Frequenzanzeige

Zeigt die aktuell eingestellte Frequenz in verschiedenen Formaten:

- **MHz:** 7.100.000 MHz (KW-Bänder)
- **kHz:** 14200 kHz
- **Hz:** 145500000 Hz (VHF/UHF)

3. S-Meter (analog oder digital)

Zeigt die Signalstärke an:

- **S1-S9:** übliche Signalstärkeskala
- **+dB:** Signale über S9 (z. B. S9+20dB)
- **Analog oder digital:** je nach Konfiguration

4. Betriebsartentasten

Wählen Sie die Demodulationsart:

- **AM:** Amplitudenmodulation
- **FM:** Frequenzmodulation
- **USB:** oberes Seitenband
- **LSB:** unteres Seitenband
- **CW:** Morsetelegrafie (Continuous Wave)
- **WBFM:** Breitband-FM (Rundfunk)
- **QUAM:** C-QUAM-AM-Stereo — die Beschriftung wird grün, sobald ein Stereopilot erkannt wird

Neben der Überschrift **Modes selector** stehen die Tasten **RADEL** und **RADEU**, die RADE v1 Digital Voice mit einem Druck starten — siehe [Digitale Decoder](#).

5. Bedienfeld

Weitere Bedienelemente:

- **AGC:** automatische Verstärkungsregelung
- **NR:** Rauschminderung (spektral)
- **NB:** Störaustaster (impulsiv)
- **NS:** Grundrauschunterdrückung
- **AN:** automatische Notch
- **CTCSS:** Subton-Rauschsperr (FM). Ein Druck öffnet neben der Taste ein kleines Fenster mit *Any tone* und den 49 Standard-Subtönen (67,0–254,1 Hz); nach der Wahl zeigt die Taste den Ton an (zum Beispiel 88.5 oder ANY), und der Ton ist nur zu hören, solange dieser Subton empfangen wird — das Öffnen dauert etwa 0,7 s. Ein erneuter Druck schaltet CTCSS aus; Escape oder ein Klick daneben schließt das Fenster, ohne etwas zu ändern. Lesezeichen speichern den gewählten Ton.
- **SQL:** Rauschsperr
- **AI:** KI-Rauschunterdrückung (nur Sprache)
- **Zoom:** Vergrößerungsstufe des Wasserfalls
- **Wheel Tuning Steps:** Schrittweite des Mausekaders
- **Decoders:** Tasten, die jeden Decoder mit einem Druck starten und stoppen

6. Bandplan-Overlay

Farbige Balken zeigen die Frequenzzuweisungen:

- unterschiedliche Farben für unterschiedliche Funkdienste
- hilft zu erkennen, was auf welcher Frequenz zulässig ist

Grundlegende Bedienung

Auf eine Frequenz abstimmen

Methode 1: In den Wasserfall klicken

1. Klicken Sie direkt auf ein Signal im Wasserfall
2. Der Empfänger stimmt auf diese Frequenz ab
3. Der Ton beginnt

Methode 2: Frequenz eingeben

1. Klicken Sie auf die Frequenzanzeige
2. Geben Sie die gewünschte Frequenz ein
3. Drücken Sie die Eingabetaste

Beispiele:

- 7100 → 7.100 MHz
- 14200.0 → 14.200 MHz
- 145.500 → 145.500 MHz

Methode 3: Mausrad verwenden

1. Bewegen Sie den Zeiger über die Frequenzanzeige
2. Rad nach oben: Frequenz erhöhen
3. Rad nach unten: Frequenz verringern

Methode 4: Schrittschaltflächen

1. Verwenden Sie die Schaltflächen ▲ und ▼ neben der Frequenz
2. Die Schrittweite hängt von der Betriebsart ab:
3. **AM/FM**: Schritte von 1 kHz
4. **SSB**: Schritte von 100 Hz
5. **CW**: Schritte von 10 Hz

Scanner

Der Scanner führt den Empfänger über einen Bereich von Kanälen und hält beim ersten an, auf dem ein Signal liegt. Er sitzt in der Zeile **Fine Tuning (kHz)** ganz rechts:



Bedienelement	Funktion
◀▶	Abwärts / aufwärts suchen. Steht der Scanner auf einem Signal, setzt ein Pfeil den Suchlauf fort.
■	Anhalten.
○	Diesen Kanal vom Suchlauf ausschließen — für Pfeifstellen oder Dauerträger.
30 dB · 4 ▼	Die Schwelle, daneben der gerade gemessene Pegel. Öffnet die Einstellungen.

Der Text links zeigt, was der Scanner tut: **Scanner** im Ruhezustand, die erreichte Frequenz während des Suchlaufs, **3s** während des Rücklaufzählers, **30·22s** solange der Kanal noch belegt ist, **hold**, wenn er stehen bleibt.

Einstellungen

- **Range** — *Scan Band* durchsucht das Band, in dem der Suchlauf beginnt; *Scan Visible* durchsucht genau den sichtbaren Wasserfall und folgt ihm beim Zoomen und Verschieben.
- **Scan** — *Every channel* stimmt jeden Kanal der Reihe nach ab und hört hinein; *Skip empty* liest das Spektrum und springt direkt zu den Signalen.
- **Stop at** — wie weit über dem Grundrauschen des Bandes ein Kanal liegen muss, damit der Suchlauf anhält, in dB. Das Rauschen wird laufend nachgeführt, eine Einstellung passt daher tags wie nachts. Ein leerer Kanal zeigt nicht 0 dB — beobachten Sie den Live-Wert auf der Schaltfläche und wählen Sie die Schwelle darüber.

- **Resume after** — wie lange ein Kanal ruhig bleiben muss, bevor der Suchlauf weiterläuft. Sprechpausen starten ihn nicht neu. *Hold* bleibt stehen, bis Sie eine Taste drücken.
- **Max stay** — läuft nach dieser Zeit weiter, auch wenn das Signal noch da ist, damit ein Dauerträger den Suchlauf nicht dauerhaft festhält.

Die Schrittweite richtet sich nach der Betriebsart — 1 kHz in SSB, 0,1 kHz in CW, 5 kHz in AM, 9 oder 10 kHz auf Mittelwelle und 9 kHz auf Langwelle — und die Haltepunkte liegen im Kanalaraster. Der Suchlauf bleibt in seinem Bereich und innerhalb dessen, was der Empfänger abstimmen kann; an einer Grenze läuft er am anderen Ende weiter.

Einstellungen und ausgeschlossene Kanäle merkt sich Ihr Browser.

Demodulationsart auswählen

Wählen Sie die zum Signal passende Betriebsart:

Für Sprechfunk:

- **AM:** Luftfahrt, AM-Rundfunk, teils Amateurfunk
- **FM:** VHF-/UHF-Relaisfunkstellen, UKW-Rundfunk
- **USB:** KW-Amateurfunk (20 m, 17 m, 15 m, 12 m, 10 m)
- **LSB:** KW-Amateurfunk (160 m, 80 m, 40 m, 30 m)

Für Daten/Digital:

- **USB:** die meisten digitalen Betriebsarten (FT8, PSK31, RTTY)
- **LSB:** einige digitale Betriebsarten auf den unteren KW-Bändern

Für Morsetelegrafie:

- **CW:** Telegrafie-/Morsesignale

Lautstärke einstellen

- **Regler auf dem Bildschirm:** Lautstärkeregler ziehen
- **Browser-Lautstärke:** Mediensteuerung des Browsers verwenden
- **Systemlautstärke:** Lautstärke des Computers anpassen
- **Tastatur:** Tasten + und - verwenden (sofern unterstützt)

Das S-Meter nutzen

Das S-Meter zeigt die Signalstärke:

- **S0-S3:** sehr schwaches Signal, schwer aufzunehmen
- **S4-S6:** schwaches bis mittleres Signal
- **S7-S9:** gutes bis starkes Signal
- **S9+:** außerordentlich starkes Signal

Tipp: Für die beste Tonqualität stimmen Sie auf Signale ab, die S7 oder mehr anzeigen.

Das Skalenbild wechseln: Klicken Sie beim analogen Instrument (Zeiger) auf das Instrument selbst — oder wählen Sie es an und drücken Sie Enter oder die Leertaste —, um drei Hintergründe durchzuschalten: dunkles gebürstetes Metall, ein helles blassgraues Zifferblatt und ein warmes Bernstein-Zifferblatt im Vintage-Stil. Ihr Browser merkt sich die Wahl, sie ist also nach einem Neuladen oder Neustart noch da. Sie gilt pro Browser und pro Adresse: Der Aufruf des Empfängers über den Hostnamen und über die IP ergibt zwei getrennte Einstellungen, und ein privates Fenster startet immer mit der Standardeinstellung der Seite.

Oberhalb von 30 MHz folgt das S-Meter dem VHF-Standard der IARU Region 1: S9 liegt dort bei -93 dBm statt -73 dBm auf Kurzwelle, weiterhin sechs dB je S-Stufe. Zeiger und Balken stellen sich mit der eingestellten Frequenz von selbst um; die Werte in dBm und dB μ V zeigen immer den gemessenen Pegel. Manche Stationen stellen ihre VHF/UHF-Empfänger außerdem so ein, dass die Instrumente **auf einem leeren Kanal auf 0 ruhen**, wie bei einem VHF/UHF-Funkgerät: Oberhalb von 60 MHz schlagen Zeiger und Balken dann erst aus, wenn ein Signal aus dem Rauschen heraustritt. Die Werte in dBm, dB μ V, SNR und NF zeigen immer die echten Werte.

Einen Empfänger wählen

Manche Stationen betreiben mehr als einen Empfänger — zum Beispiel einen für Kurzwelle und einen für das 2-m-Band. Sie zeigen im Seitenkopf eine Zeile *Receivers*: mit einer Schaltfläche pro Empfänger; der, auf dem Sie gerade sind, erscheint gelb. Ein Druck auf eine andere Schaltfläche öffnet die Seite dieses Empfängers, mit eigenem Wasserfall, eigenen Bändern, Chat und eigener Hörerliste. Auf der Seite / mobile sitzen dieselben Schaltflächen in der oberen Leiste. Eine Station mit nur einem Empfänger zeigt keine solche Zeile.

Erweiterte Funktionen

Automatische Verstärkungsregelung (AGC)

Die AGC passt die Audiopegel automatisch an:

- **Off**: keine automatische Verstärkungsanpassung
- **Slow**: allmähliche Pegeländerungen (am besten für SSB)
- **Medium**: ausgewogenes Verhalten
- **Fast**: schnelle Anpassung (am besten für AM)

Empfehlung: Beginnen Sie mit „Fast“ für AM und „Slow“ für SSB.

Die vier Rauschregler

NR, NB, NS und AN sind getrennte Ein/Aus-Schalter, die jeweils eine andere Art von Störung angehen. Sie sind unabhängig voneinander — einer schaltet keinen anderen mit ein — und lassen sich beliebig kombinieren.

Keiner von ihnen erreicht die Decoder: FT8, CW, WSPR, SSTV, FAX, NAVTEX, RTTY/PSK31/Olivia und der QRSS-Grabber lesen das Audio *vor* diesen Filtern. Sie können sie also rein nach Gehör einstellen, ohne zu beeinflussen, was decodiert wird. Siehe das [Decoder-Handbuch](#).

Rauschminderung (NR)

Spektrale Rauschminderung. Sie schätzt den Rauschpegel in jedem Teil des Audiospektrums und dreht diese Anteile herunter; was über dem Rauschen steht, bleibt unangetastet.

Verwenden, wenn: Sie gleichmäßiges Zischen oder weißes Rauschen hinter dem Signal hören.

Dauertöne — eine CW-Note, ein Träger — werden als Signal erkannt und geschützt. NR frisst also kein CW-Signal, wie es ein naives Filter täte. Typische Wirkung: 10 dB weniger Rauschen in den Sprechpausen, für ein paar Zehntel dB Verlust an der Sprache selbst.

Störaustaster (NB)

Entfernt Impulsstörungen: Knacken, Knallen, atmosphärische Entladungen, Zünd- und Netzstörungen. Er beobachtet die Audio-Hüllkurve und schweigt nur die Abtastwerte still, die weit darüber ausschlagen — etwa eine Millisekunde je Knall, mit weichen Flanken, damit die Austastung nicht selbst knackt.

Verwenden, wenn: Sie Knacken von Stromleitungen, Motoren, Gewittern oder Fahrzeugzündungen hören.

Die Netzbrumm-Notches bei 50 Hz und 60 Hz folgen dieser Taste.

Grundrauschunterdrückung (NS)

Misst über mehrere Sekunden das Grundrauschen des Bandes und senkt alles ab, was auf diesem Pegel liegt — bei hohen Tonfrequenzen stärker als bei tiefen. Wo NR von Moment zu Moment reagiert, ist NS die langsame, ruhige Hand: Sie senkt das Bandrauschen, ohne den Klang des Signals selbst zu verändern.

Verwenden, wenn: Das Band ruhig, aber zischig ist und Sie das Rauschen absenken wollen, ohne den „Unterwasser“-Klang aggressiver NR.

Nur in USB, LSB und AM aktiv — CW, FM und die digitalen Betriebsarten bleiben unberührt. Bei jedem Frequenz- oder Modewechsel wird neu gemessen; das ist nach wenigen Sekunden eingeschwungen.

Automatische Notch (AN)

Findet und entfernt selbsttätig Dauertöne — Heterodyne, Träger, Pfeifstellen — ohne dass Sie ein Notchfilter von Hand setzen müssen. Sie passt sich laufend an, kann also mehrere Töne gleichzeitig entfernen und bleibt auch an einem driftenden Ton hängen.

Verwenden, wenn: Sie einen Pfeifton über dem gewünschten Signal hören.

In CW abgeschaltet, denn dort *ist* das Nutzsignal ein Dauerton.

Hinweis zur Verzögerung: NR und NS fügen im eingeschalteten Zustand je rund 40 ms Audioverzögerung hinzu (beide zusammen etwa 80 ms). NB und AN fügen keine hinzu. Das betrifft nur das Hören, nie die Decoder.

Automatische Rauschsperr (SQL)

Schaltet den Ton stumm, wenn kein Signal anliegt:

- **Off:** immer hörbar (Rauschen zu hören)
- **Auto:** Schwelle wird automatisch gesetzt
- **Manual:** Schwelle manuell einstellen

Verwenden, wenn: Sie eine Frequenz auf Aktivität überwachen.

KI-Rauschunterdrückung (AI)

Die Taste **AI** mit einem Stärkeregler daneben sitzt im Bereich Audio & Buffer direkt unter SQ. Auf der Seite /mobile befindet sie sich im Reiter Audio unter Squelch. Sie entfernt Bandrauschen aus Sprache mit RNNoise, einem kleinen, auf Sprache trainierten neuronalen Netz.

Alles geschieht in Ihrem eigenen Browser: Nichts wird an einen fremden Server geschickt, es gibt kein Konto, und der Empfänger hat keine zusätzliche Arbeit.

- **Ein/Aus: AI** anklicken; die Taste wird blau. Beim ersten Klick wird das Modul geladen (etwa 1,3 MB), und die Taste pulsiert, solange es lädt.
- **Stärke:** Der Regler bestimmt, wie stark sie wirkt, während jemand spricht — niedrigere Werte behalten mehr vom ursprünglichen Klang. Das obere Ende der Skala ist absichtlich weicher als ein voller Schnitt, damit die Sprache selbst bei 100 % nicht beschädigt wird; 100 % reinigt trotzdem mehr als 50 %. Zwischen den Wörtern arbeitet sie von sich aus stärker, sodass das Rauschen dort weiter sinkt. Voreinstellung ist 50 %. Sie können den Wert auch bei ausgeschalteter KI setzen.

- **Statusanzeige:** Die Anzeige AI unter der Frequenz leuchtet cyan, solange AI arbeitet, und erscheint halb abgedunkelt, wenn AI eingeschaltet ist, die aktuelle Betriebsart aber keine Sprachbetriebsart ist.

Verwenden, wenn: Sie SSB- oder AM-Sprache auf einem verrauschten Band hören. Das Rauschen zwischen den Worten sinkt bei der Voreinstellung 50 % typisch um etwa 10 dB, bei 100 % um mehr, bis etwa 20 dB, während die Sprache ihren Pegel behält.

Nur Sprachbetriebsarten — USB, LSB, AM und SAM. In CW, FM, den Datenbetriebsarten und C-QUAM läuft der Ton unverändert durch, weil das Netz einen CW-Ton oder Musik als Rauschen behandelt. Wie die vier Rauschregler erreicht sie nie die Decoder. Bei schwachen Stationen (unter etwa +6 dB SNR) misst sie den Signal-Rausch-Abstand selbst und glättet ihre Entscheidungen stärker, damit die Sprache nicht den blubbernden, „wässrigen“ Klang annimmt, den ein solches Filter erzeugen kann; starke Stationen werden genau wie bisher verarbeitet. Klingt ein sehr schwaches Signal trotzdem bearbeitet, senken Sie den Regler. Sie fügt etwa 50 ms Tonverzögerung hinzu.

Audiopuffer und Verzögerung

Der Ton erreicht den Browser in kleinen Paketen, bei einem typischen Empfänger je 35 ms, und der Browser hält ein wenig davon vorrätig, damit ein verspätetes Paket keine Lücke verursacht. Dieser Vorrat ist der **Puffer**, und er macht den größten Teil der Verzögerung zwischen Empfänger und Lautsprecher aus.

- **Desktop:** Im Bereich Audio & Buffer erhöht die Taste mit dem Netzwerksymbol den Puffer pro Klick um eine Stufe (sie wird oberhalb der niedrigsten Stufe grün), und der Schieberegler daneben wählt direkt eine Stufe von 1 bis 6. Die niedrigste Stufe passt zu einer Kabelverbindung; Telefone beginnen auf Stufe 2.
- **/mobile:** Die Taste **BUF** und der Regler **Buffer** im Reiter Audio tun dasselbe.

Der Puffer passt sich selbst an. Die Seite misst, wie unregelmäßig Ihre Verbindung den Ton liefert, und hält nur so viel vorrätig, wie die letzte Minute gezeigt hat, plus die Sicherheitsreserve der gewählten Stufe: 20 ms auf Stufe 1, 50 ms auf Stufe 2, mehr auf den höheren Stufen. Nach einer kurzen Netzstörung baut sich die zusätzliche Verzögerung innerhalb von ein, zwei Minuten wieder ab, und ein kleiner Unterschied zwischen dem Takt Ihrer Soundkarte und dem des Empfängers führt nicht mehr alle paar Minuten zu einer Unterbrechung. Die Anpassungen betragen jeweils wenige Millisekunden, erfolgen dort, wo sich der Klang wiederholt, und sind nicht hörbar. Decoder und Ihre Aufnahmen sind nie betroffen: Sie erhalten jedes Sample.

Bricht der Ton ab, erhöhen Sie den Puffer um eine Stufe. Auf einem Telefon im heimischen WLAN beträgt die Verzögerung vom Empfänger bis zum Lautsprecher typisch etwa 100–150 ms; das Netz zwischen Ihnen und dem Empfänger kommt hinzu.

Chat

Hat der Betreiber den Chat aktiviert, hat er auf der Desktop-Seite einen eigenen Bereich und auf /mobile einen Reiter **Chat**. Geben Sie Ihren Namen oder Ihr Rufzeichen ein, schreiben Sie Ihre Nachricht und senden Sie sie.

Um auf eine bestimmte Nachricht zu antworten, klicken Sie auf ➡ **Reply** neben ihrer Uhrzeit. Eine Zeile über dem Eingabefeld zeigt, wem und worauf Sie antworten; schreiben und senden Sie wie gewohnt, oder drücken Sie × bzw. Esc zum Abbrechen. Die Antwort erscheint eingerückt unter der beantworteten Nachricht, mit einer kurzen Zitatzeile. Eine Antwort auf eine Antwort gehört zum selben Faden. Der Server behält die letzten 20 Nachrichten; ist das Original schon aus diesem Verlauf verschwunden, zeigt eine Antwort nur Name und Uhrzeit dessen, worauf sie antwortet.

Zoomfunktion

Vergrößert die Wasserfallanzeige:

- **1x**: Normalansicht (breite Abdeckung)
- **2x**: 2-fache Vergrößerung
- **4x**: 4-fache Vergrößerung
- **8x**: 8-fache Vergrößerung

Verwenden, wenn: Sie Signale deutlicher sehen oder genauer abstimmen möchten.

Transceiver-Steuerung (CAT)

Ihr eigener Transceiver und die Empfängerseite lassen sich auf derselben Frequenz halten: Drehen Sie am Abstimmknopf des Funkgeräts, folgt der Wasserfall – oder klicken Sie auf ein Signal im Wasserfall, und das Funkgerät stimmt darauf ab. Das betrifft nur *Ihre* eigene Hörsitzung; niemand sonst am Empfänger merkt etwas davon.

Mit Desktop PhantomSDR+ (ab 4.0). Die Desktop-Anwendung hat ein Menü **Rig**. *Rig* → *Rig control...* öffnet ein Fenster, in dem Sie Ihr Funkgerät und den Verbindungsweg wählen; das Menü selbst schaltet die Synchronisierung ein und aus. Synchronisiert werden Frequenz, Betriebsart und Filterbreite, in eine Richtung oder in beide, und auf Wunsch wird der Empfänger während des Sendens stummgeschaltet. Neben PhantomSDR-Plus steuert sie auf dieselbe Weise auch Empfänger **KiwiSDR**, **PA3FWM WebSDR** und **UberSDR**. Das Funkgerät wird auf einem von vier Wegen erreicht:

- **Integriert** – ohne weitere Software: Icom (CI-V), Yaesu (neues CAT sowie FT-817/857/897), Kenwood, Elecraft, FlexRadio SmartSDR CAT, QRP Labs und andere Kenwood-kompatible Geräte.
- **Hamlib** – jedes Funkgerät, das Hamlib kennt, mehr als 300, ausgewählt aus einer durchsuchbaren Liste. Die Windows-Installer enthalten Hamlib; unter Linux installieren Sie `libhamlib-utils`.
- **rigctld im Netzwerk** – ein bereits laufender `rigctld`.
- **flrig** – für ein Funkgerät, das flrig schon mit fldigi, WSJT-X oder einem Logbuch teilt.

Die Anwendung, ihre Installer und das vollständige Handbuch finden Sie unter [Desktop PhantomSDR+ \(Dropbox\)](#).

Mit einem Webbrowser. Das [CATsync Tool for WebSDRs](#) (Windows) koppelt ein Funkgerät an die im Browser geöffnete Empfängerseite. Es synchronisiert Frequenz und Betriebsart.

Filterbreite und Stummschaltung beim Senden funktionieren mit KiwiSDR-, WebSDR- und UberSDR-Empfängern sowie mit einem PhantomSDR-Plus-Empfänger mit 4.1.0 oder neuer. Bei einem älteren PhantomSDR-Plus werden Frequenz und Betriebsart weiterhin synchronisiert, der Filter nicht.

Das vollständige Handbuch — jede Einstellung, die unterstützten Geräte, wie verhindert wird, dass sich beide Seiten bekämpfen, und die Fehlerbehebung — ist **Transceiver-Steuerung**.

Für Entwickler. Jede Empfängerseite stellt diese Funktionen auf `window` bereit; beide Werkzeuge nutzen sie:

Funktion	Wirkung
<code>catsync_getFrequency()</code> / <code>catsync_setFrequency(hz)</code>	Abgestimmte Frequenz in Hz
<code>catsync_getMode()</code> / <code>catsync_setMode(mode)</code>	USB, LSB, CW, CW-L, AM, FM, WBFM ...
<code>catsync_getBandwidth()</code> / <code>catsync_setBandwidth(hz)</code>	Gesamte Durchlassbreite in Hz. Nach der Betriebsart setzen: ein Wechsel der Betriebsart setzt den Durchlassbereich zurück
<code>catsync_getMute()</code> / <code>catsync_setMute(on)</code>	Stummschaltung, über die Stummschalttaste der Seite
<code>catsync_ready</code>	<code>true</code> , sobald die Funktionen installiert sind

Die letzten drei Zeilen sind mit 4.1.0 hinzugekommen; prüfen Sie vor dem Aufruf, ob eine Funktion vorhanden ist.

Demodulationsarten

AM (Amplitudenmodulation)

Verwendet für:

- Flugfunk
- AM-Rundfunk
- Teile des Amateurfunks
- Seefunk

Eigenschaften:

- große Bandbreite (typisch 10 kHz)
- stör anfällig
- leicht abzustimmen (einfach auf das Signal klicken)

Bewährte Praxis:

- schnelle AGC verwenden
- Störaustaster einschalten, wenn Sie Knacken hören
- auf das Maximum des Signals im Wasserfall abstimmen

QUAM (C-QUAM-AM-Stereo)

Verwendet für:

- Mittelwellensender, die AM-Stereo ausstrahlen

Eigenschaften:

- 10 kHz breit, wird als Stereopaar statt als Mono-AM decodiert
- Die Tastenbeschriftung wird von selbst **grün**, sobald der 25-Hz-Stereopilot anliegt — so sehen Sie schon vor dem Umschalten, welche Sender wirklich in Stereo senden
- Aus AM heraus schaltet ein weiterer Druck auf synchrone AM-Demodulation am Träger um; die Beschriftung wird **gelb** und zeigt SAM. Ein dritter Druck führt zurück zu einfachem AM
- C-QUAM-Audio wird mit Opus übertragen, alle anderen Betriebsarten nutzen FLAC

Bewährte Vorgehensweise:

- Nachts auf starken Mittelwellensignalen nach der grünen Beschriftung suchen
- Klingt Stereo instabil, ist SAM bei schwachem Träger die ruhigere Wahl

FM (Frequenzmodulation)**Verwendet für:**

- VHF-/UHF-Amateurfunkrelais
- Behördenfunk (Polizei, Feuerwehr, Rettungsdienst)
- kommerziellen Betriebsfunk
- einige Satellitenverbindungen

Eigenschaften:

- schmale Bandbreite (typisch 12,5 oder 25 kHz)
- ausgezeichnete Störfestigkeit
- „Mitnahmeeffekt“ (das stärkste Signal setzt sich durch)

Bewährte Praxis:

- genau auf die Signalmitte abstimmen
- Rauschsperr nutzen, um im Ruhezustand stummzuschalten
- Rauschminderung ausschalten (nicht nötig)

USB (oberes Seitenband)**Verwendet für:**

- KW-Amateurfunk (oberhalb 10 MHz)
- die meisten digitalen KW-Betriebsarten
- Seefunk (oberhalb 8 MHz)

Eigenschaften:

- schmale Bandbreite (typisch 2.4 kHz)
- effiziente Spektrumnutzung
- erfordert genaue Abstimmung

Bewährte Praxis:

- langsame AGC verwenden
- auf die untere Signalkante im Wasserfall abstimmen
- bei Bedarf Rauschminderung einschalten

LSB (unteres Seitenband)**Verwendet für:**

- KW-Amateurfunk (unterhalb 10 MHz)
- einige digitale KW-Betriebsarten
- Seefunk (unterhalb 8 MHz)

Eigenschaften:

- wie USB, aber spiegelbildlich
- Konvention: LSB auf den unteren KW-Bändern

Bewährte Praxis:

- langsame AGC verwenden
- auf die obere Signalkante im Wasserfall abstimmen
- bei Bedarf Rauschminderung einschalten

CW (Continuous Wave / Morsetelegrafie)**Verwendet für:**

- Amateurfunktelegrafie
- Navigationsbaken
- Zeitzeichensender

Eigenschaften:

- sehr schmale Bandbreite (100-500 Hz)
- hohe Effizienz
- erfordert Kenntnis des Morsecodes

Bewährte Praxis:

- schmales Filter verwenden (400-500 Hz)
- genau auf die Tonmitte abstimmen
- Audiofilter für einen besseren Ton einschalten

CW-L (CW, unteres Seitenband)

Derselbe ± 250 -Hz-Morsefilter wie **CW**, der Ton wird aber unterhalb statt oberhalb des Trägers entnommen. Nützlich, wenn ein Signal auf der unteren Seite besser aufzunehmen ist oder ein störender Träger direkt über dem gewünschten liegt.

Auf der Desktop-Seite gibt es keine CW-L-Taste. Deren Betriebsartenreihe lautet USB · LSB · CW · AM · QUAM · FM. CW-L wird auf der vereinfachten Seite /mobile angeboten; am Desktop lässt es sich weiterhin über einen Link oder ein Lesezeichen setzen, das es benennt.

WBFM (Breitband-FM)**Verwendet für:**

- UKW-Rundfunk (88-108 MHz)
- einige Satelliten-Downlinks

Eigenschaften:

- sehr große Bandbreite (200 kHz)
- ausgezeichnete Tonqualität
- hohe Wiedergabetreue

Bewährte Praxis:

- genau auf die Mittenfrequenz abstimmen
- für Rundfunk keine Rauschsperrung nötig
- genießen Sie den hochwertigen Klang!

Digitale Decoder

PhantomSDR-Plus enthält integrierte Decoder für digitale Betriebsarten. Eine vollständige Anleitung finden Sie unter [Decoder](#).

Der schnellste Weg ist die Tastenreihe **Decoders** im Hauptpanel, direkt unter **Wheel Tuning Steps**: zehn Tasten — **FT8, FT4, FT2, JS8, CW, WSPR, FAX, SSTV, NAVTX, RTTY** —, die mit einem einzigen Druck ihren Decoder starten, den Hauptschalter Decoder auf ON stellen und das Fenster des Decoders in den sichtbaren Bereich holen. Die Taste bleibt blau, solange der Decoder läuft; ein erneuter Druck stoppt den Decoder und schließt sein Fenster. Das Dropdown-Menü **Decoder Options** funktioniert weiterhin genau wie bisher und bleibt mit den Tasten synchron.

RADEL und **RADEU** (RADE v1 Digital Voice) haben eigene Tasten neben der Überschrift **Modes selector** sowie in den Popup-Fenstern **Modes** und **Bands** und arbeiten nach demselben Prinzip: drücken zum Starten, erneut drücken zum Stoppen.

Die Decoder für SSTV, HF-FAX, NAVTEX, FSK/RTTY und CW laufen jeweils in einem eigenen Hintergrund-Thread, sodass das Ein- oder Ausschalten eines Decoders den Ton nie unterbricht und der Wasserfall während des Dekodierens flüssig bleibt. Sie erhalten das Audiosignal vor AGC, Rauschminderung und Stummschaltung – Sie können den Empfänger stummschalten, und die Dekodierung läuft unbeeinflusst weiter.

Solange ein Decoder läuft, hält er Betriebsart und Durchlassbereich. Normalerweise folgt die Betriebsart dem Bandplan — stimmen Sie in einen als LSB oder AM markierten Abschnitt, schaltet der Empfänger um. Ein laufender Decoder setzt sich darüber hinweg und behält die Betriebsart und den schmalen Durchlassbereich, die er braucht, selbst wenn Sie auf ein anderes Band gehen. Die bandeigene Betriebsart kehrt zurück, sobald Sie den Decoder abschalten, und die Mode-Schaltflächen greifen jederzeit, wenn Sie übernehmen wollen. Der CW-Decoder ist die Ausnahme: Er dekodiert in der Betriebsart, in der Sie gerade hören.

FT8-, FT4-Decoder

Was ist FT8, FT4?

- weit verbreitete digitale Amateurfunk-Betriebsart
- Verbindungen mit schwachen Signalen
- 15-Sekunden-Aussendungen bei FT8, 7,5-Sekunden-Aussendungen bei FT4

Anwendung:

1. Stimmen Sie auf FT8- oder FT4-Frequenzen ab
2. Wählen Sie die Betriebsart USB
3. Aktivieren Sie den FT8-Decoder — drücken Sie die Taste **FT8** oder wählen Sie ihn über das Menü
4. Beobachten Sie, wie die dekodierten Meldungen erscheinen

Übliche FT8-Frequenzen (USB):

- 160 Meter: 1.840 MHz
- 80 Meter: 3.573 MHz
- 60 Meter: 5.357 MHz
- 40 Meter: 7.074 MHz
- 30 Meter: 10.136 MHz
- 20 Meter: 14.074 MHz
- 17 Meter: 18.100 MHz
- 15 Meter: 21.074 MHz

- 12 Meter: 24.915 MHz
- 10 Meter: 28.074 MHz
- 6 Meter: 50.313 MHz (50.323 MHz für DX)

Übliche FT4-Frequenzen (USB):

- 80 m: 3.575 MHz
 - 40 m: 7.0475 MHz
 - 30 m: 10.140 MHz
 - 20 m: 14.080 MHz
 - 17 m: 18.104 MHz
 - 15 m: 21.140 MHz
 - 12 m: 24.919 MHz
 - 10 m: 28.180 MHz
 - 6 m: 50.318 MHz
-

JS8-Decoder

Was ist JS8?

- FT8s Schwachsignal-Engine, genutzt für Gespräche von Tastatur zu Tastatur
- Freier Text statt fester Meldungen: lange Nachrichten kommen über mehrere Zyklen
- Fünf Geschwindigkeiten; **Normal** (15 s) ist die Anruf-Geschwindigkeit und trägt fast allen Verkehr

Bedienung:

1. Auf eine JS8-Frequenz abstimmen
2. **USB** wählen
3. Den JS8-Decoder einschalten — Taste **JS8** oder Auswahl im Menü
4. **Speed** auf *Normal* und den Sync-Regler auf **Auto** lassen

Das Fenster lesen:

- Noch eintreffende Nachrichten stehen oben in Grün mit blinkendem Cursor — bei Normal kann eine Nachricht eine volle Minute brauchen
- Fertige Nachrichten darunter als Mode | Hz | dB | Message, **Rufzeichen in Grün**
- Eine blasse, kursive Zeile lief ab, bevor der letzte Rahmen ankam: der Text ist echt, kann aber abgeschnitten sein

Übliche JS8-Frequenzen (USB):

- 160m: 1.842 MHz
- 80m: 3.578 MHz
- 40m: 7.078 MHz
- 30m: 10.130 MHz
- 20m: 14.078 MHz
- 17m: 18.104 MHz
- 15m: 21.078 MHz
- 12m: 24.922 MHz
- 10m: 28.078 MHz

JS8 ist viel ruhiger als FT8 — eine Aussendung alle paar Minuten ist normal, Pausen sind kein Fehler. Vollständige Anleitung: [Decoder](#).

CW-Decoder

Drücken Sie einfach die CW-Schaltfläche, und der dekodierte Text erscheint. Drücken Sie die CW-Schaltfläche erneut, um das Fenster zu leeren und den Decoder neu zu starten.

WSPR

WSPR (Weak Signal Propagation Reporter, ausgesprochen „whisper“) ist eine Bakenbetriebsart für extrem schwache Signale, die KW-Ausbreitungswege weltweit kartiert. Jede Aussendung dauert etwa 110 Sekunden und passt in einen 200 Hz breiten Kanal. Der Decoder wartet auf einen vollständigen, an UTC ausgerichteten 2-Minuten-Zeitschlitz, bevor er dekodiert. Aktivieren Sie den Decoder und wählen Sie **WSPR** aus der Auswahlliste. Oder drücken Sie einfach die Taste **WSPR**.

HF-FAX / WEFAX

HF-Radiofax (auch WEFAX genannt) wird von Küstenwachen und Wetterdiensten weltweit genutzt, um Wetterkarten, Seegangskarten und Bodenanalysen über Kurzwelle auszustrahlen. Der Decoder setzt das Bild Zeile für Zeile zusammen, während es empfangen wird.

NAVTEX

NAVTEX ist das internationale seefunkgestützte Aussendungssystem für küstennahe Sicherheitsinformationen – Navigationswarnungen, Wettervorhersagen sowie Such- und Rettungsmeldungen.

FSK / RTTY, PSK31 und Olivia

Ein universeller Decoder für schmalbandige Textbetriebsarten mit fünf Varianten in einem Fenster: maritimes FSK (SITOR), Wetter-RTTY, Amateur-RTTY, **PSK31** (Phasenumtastung, 31,25 Baud) und **Olivia** (Mehrton-FSK mit Vorwärtsfehlerkorrektur). Das Bedienfeld passt sich der Variante an — die Regler für Shift, Baud und Rahmung verschwinden bei PSK31 und Olivia, und Olivia ergänzt einen Mode-Wähler und einen Squelch-Regler.

Amateur-RTTY und Wetter-RTTY haben einen Regler **Squelch (SNR)**, der verhindert, dass Rauschen als zufällige Buchstaben ausgegeben wird, und alle Varianten außer Olivia zeigen eine echte SNR auf derselben 3-kHz-Skala. Zwei Dinge sind wichtig: PSK31 korrigiert seinen Abstimmfehler über etwa ± 25 Hz selbst, Sie müssen also nur in die Nähe kommen; Olivia braucht einen **Mode** (Töne / Bandbreite), der exakt zur Aussendung passt, sendet keine Präambel und benötigt daher einige Sekunden zur Synchronisierung, bevor Text erscheint. Das Bedienfeld startet mit Olivia **8 / 250**.

QRSS-Grabber

QRSS ist CW, das so langsam gesendet wird, dass ein einzelner Punkt Sekunden dauert; es wird angesehen statt gehört — der Grabber zeichnet die Spur auf seiner eigenen Anzeige. Er steht nicht im Decoder-Auswahlmenü, sondern hat einen eigenen Bereich **QRSS** und kann gleichzeitig mit einem Decoder laufen.

Drücken Sie **Show**, wählen Sie dann ein Fenster aus der Liste **Band** und drücken Sie **Tune**: Der Empfänger geht in **CW** auf diese Frequenz, mit einem für die Betriebsart passenden Durchlassbereich,

und die Spur landet auf der Mittellinie der Anzeige. 30 m (10.140,00 kHz) ist das belebteste Fenster. Stellen Sie **Speed** passend zur Bake ein — **QRSS 10**, wenn Sie es nicht wissen — und haben Sie Geduld: Ein Rufzeichen kann zehn Minuten brauchen, um über den Schirm zu wandern. Einzelheiten in [Decoder](#).

SSTV

Slow-Scan-Television überträgt Standbilder Zeile für Zeile über einen gewöhnlichen SSB-Kanal. Aktivieren Sie den Decoder, wählen Sie **SSTV** und stimmen Sie auf eine SSTV-Frequenz ab – 14.230 MHz ist die wichtigste internationale Anruffrequenz. Belassen Sie **Mode** auf **Auto**: Der Decoder liest den VIS-Kopf der Aussendung und bestimmt die Betriebsart notfalls anhand des Synchronisationstimmings, falls Sie erst nach dem Kopf eingestiegen sind. Die Modi Martin, Scottie und Robot werden unterstützt, und das Bild baut sich Zeile für Zeile auf. Oder drücken Sie einfach die Taste **SSTV**.

Automatisches Spot-Reporting und Systemgraphen

FT8-, FT4- und WSPR-Dekodierungen kann auch der Server selbst hochladen — FT8/FT4 an PSK Reporter, WSPR an WSPRnet — über einen Autorun-Daemon, den der Sysop im Admin-Panel startet. Er ist von den Decodern in Ihrem Browser unabhängig: er läuft weiter, ob jemand zuhört oder nicht, und nichts, was Sie im Browser dekodieren, wird gemeldet.

Der Sysop verfolgt ihn über zwei Zähler, die leicht zu verwechseln sind: Die Kacheln je Decoder zählen die seit dem letzten Start des Daemons hochgeladenen Spots, während die Zahl neben dem Kontrollkästchen jedes Bands bzw. jeder Betriebsart der Gesamtwert seit Beginn ist und Neustarts übersteht. Dasselbe Panel enthält eine Seite **Graphen**, die CPU-Takt, CPU-Last, CPU-Temperatur und Nutzer online über die letzten 15 Minuten bis 24 Stunden darstellt. Beides beschreibt das [Admin-Panel-Handbuch](#).

Tastenkürzel

Tastenkürzel für zügigeres Arbeiten.

Frequenzsteuerung

Die Frequenzanzeige ist eine Ziffernreihe, die Sie direkt bedienen. **Klicken Sie zuerst auf eine Ziffer** — das wählt sie aus, und alles Folgende wirkt auf die Auswahl.

- **Pfeil links / rechts**: verschiebt die Auswahl zur nächsten Ziffer (acht Ziffern, 100 MHz bis 10 Hz)
- **Pfeil hoch / runter**: erhöht oder verringert die *ausgewählte* Ziffer um ihren Stellenwert — auf der MHz-Ziffer also 1 MHz, auf der letzten 10 Hz
- **0 - 9**: schreibt die Ziffer direkt an die ausgewählte Stelle

Mausrad über der Frequenzanzeige: schrittet mit der Abstimmungsschrittweite des Bandes (1 kHz, sofern der Bandplan nichts anderes vorgibt). **Shift** für 1 kHz, **Alt** für 10 kHz.

Mausrad über dem Wasserfall: zoomt. Mit **Ctrl** (oder **Cmd**) bzw. **Shift** wird stattdessen abgestimmt, mit **Shift + Ctrl** auf volle kHz gerundet.

Page-Up- / Page-Down-Kürzel gibt es nicht.

Lesezeichen

Speichern Sie Ihre Lieblingsfrequenzen für den schnellen Zugriff. Sie können die Lesezeichenliste auch exportieren, lokal speichern und anschließend in ein beliebiges anderes PhantomSDR importieren.

Wenn sich Lesezeichen und Marker überlappen:

Blaue Lesezeichen erscheinen oben

Gelbe Marker erscheinen darunter

Klicks auf Lesezeichen haben Vorrang

Ein Lesezeichen hinzufügen

1. Auf die gewünschte Frequenz abstimmen
2. Auf die Schaltfläche „Bookmarks" klicken
3. Auf „Add Bookmark" klicken
4. Eine Beschreibung eingeben
5. Auf „Save" klicken

			Bookmark name Label (optional) [Add]
--	--	--	--

Verwendung:

Lesezeichen hinzufügen:

- Name: „Local News Station"
- Label: „NEWS"
- Auf Add klicken

Anzeige im Wasserfall:

- hineinzoomen, bis die Marker erscheinen
- ein marineblauer Kasten mit „NEWS" in fettem Gelb erscheint

Lesezeichen anklicken:

- stimmt auf die Frequenz ab
- stellt die Demodulationsart ein
- funktioniert genau wie ein Klick auf einen Marker

Lesezeichen verwalten

- **Bearbeiten:** auf das Stiftsymbol neben dem Lesezeichen klicken
- **Löschen:** auf das Papierkorbsymbol neben dem Lesezeichen klicken
- **Exportieren:** Lesezeichen als JSON-Datei herunterladen
- **Importieren:** Lesezeichen aus einer JSON-Datei hochladen

Lesezeichen weitergeben

1. Auf „Export Bookmarks" klicken
2. Die JSON-Datei an andere weitergeben
3. Die Empfänger klicken auf „Import Bookmarks"
4. Sie wählen Ihre Datei aus

Nutzung auf Mobilgeräten

PhantomSDR-Plus funktioniert hervorragend auf Mobilgeräten!

Zwei mobile Ansichten

Es gibt zwei Wege, den Empfänger auf dem Telefon zu benutzen:

- **`http://ihr_server:PORT/mobile`** — die vereinfachte Seite. Kein Wasserfall, dadurch etwa halb so viel Datenverbrauch. Frequenzeingabe, Abstimmungsschritte, Betriebsarten, S-Meter, Bänder, Lesezeichen, Benutzerliste und Chat.
- **Die erweiterte Ansicht** — das Telefon-Layout der Hauptoberfläche, mit Wasserfall und dem vollen Umfang der Bedienelemente.

Gewechselt wird mit den Schaltflächen am unteren Rand von `/mobile` (**Mobile extended view, Full desktop view**) und mit **Simplified mobile** in der erweiterten Ansicht.

Ihre Frequenz kommt mit. Ein Ansichtswechsel lässt Sie auf demselben Signal — Frequenz und Betriebsart reisen im Link mit, Sie landen also nicht mehr auf der Standardfrequenz des Empfängers. Auch die Adresszeile folgt Ihrer Abstimmung: Neuladen, ein Lesezeichen oder ein an jemanden verschickter Link führen genau auf diese Frequenz zurück.

Die Betriebsart folgt auf beiden Seiten dem Bandplan. Stimmen Sie in ein Segment ab, das als AM, LSB, USB oder CW eingetragen ist — durch Eingeben einer Frequenz, schrittweises Abstimmen oder eine Bandtaste — schaltet der Empfänger auf diese Betriebsart um, auf der vereinfachten Seite ebenso wie in der vollen Oberfläche. Eine von Hand gewählte Betriebsart bleibt bestehen, solange Sie sich innerhalb desselben Segments bewegen; außerhalb der definierten Bänder wird Ihre Betriebsart nicht angetastet. Während **RADE** (RADEL/RADEU) läuft, behält es den Empfänger, sodass das Abstimmen den Decoder nicht unterbricht.

Beim Wechsel zwischen den beiden Ansichten kommt Ihre aktuelle Betriebsart mit; kennt die Zielansicht sie nicht, entscheidet der Bandplan für diese Frequenz — eine Rundfunkfrequenz kommt in AM an, 40 m in LSB, ein CW-Segment in CW. **SAM** auf der vereinfachten Seite wird in der erweiterten Ansicht zu AM mit dem Synchrondemodulator und umgekehrt. **RADEL** und **RADEU** gibt es nur auf der vereinfachten Seite; wechseln Sie von dort weg, bleibt die Frequenz erhalten und der Bandplan wählt die Betriebsart.

Eine Frequenz außerhalb des Empfangsbereichs wird auf die nächste Bereichsgrenze gezogen, ein alter Link kann Sie also nie außerhalb des Bandes stranden lassen.

Besonderheiten auf Mobilgeräten

- **Fingerfreundliche Bedienelemente:** große Schaltflächen und Regler
- **Wischen zum Abstimmen:** nach links/rechts über den Wasserfall wischen
- **Zwei-Finger-Zoom:** den Wasserfall zum Vergrößern/Verkleinern aufziehen
- **Querformat:** für eine bessere Ansicht drehen

Tipps für Mobilgeräte

1. **WLAN nutzen:** Audiostreaming verbraucht Datenvolumen
2. **Querformat:** bessere Sicht auf den Wasserfall
3. **Kopfhörer:** bessere Tonqualität
4. **Favoriten als Lesezeichen:** erleichtert das Wiederfinden von Stationen
5. **Andere Apps schließen:** sorgt für flüssigen Betrieb

Empfohlene mobile Browser

- **Android:** Chrome oder Samsung Internet

- **iOS:** Mozilla
 - **Beide:** achten Sie darauf, dass der Browser aktuell ist
-

Tipps und bewährte Praktiken

Für besten Empfang

1. **Starke Signale wählen:** achten Sie auf Orange/Rot im Wasserfall
2. **Genau abstimmen:** direkt auf die Signalmitte klicken
3. **Richtige Betriebsart wählen:** passend zum Signaltyp
4. **AGC einstellen:** schnell für AM, langsam für SSB
5. **NR/NB nutzen:** hilfreich bei störrischen Bedingungen

Aktivität finden

1. **Den Wasserfall beobachten:** die Farben zeigen die Signalstärke
2. **Auf beliebten Frequenzen hören:**
3. 40 m: 7.100-7.300 MHz (LSB)
4. 20 m: 14.200-14.350 MHz (USB)
5. 2 m: 145.200-145.600 MHz (FM)
6. **Bandplan-Overlay prüfen:** zeigt die Frequenzzuweisungen
7. **Lesezeichen nutzen:** schneller Zugriff auf aktive Frequenzen

Ausbreitungsbedingungen verstehen

Kurzwelle tagsüber:

- die höheren Bänder funktionieren besser (20 m, 15 m, 10 m)
- Weitverbindungen (DX) sind möglich
- Rundfunkstationen sind hörbar

Kurzwelle nachts:

- die niedrigeren Bänder funktionieren besser (80 m, 40 m)
- andere Ausbreitungsmuster
- andere Stationen sind hörbar

VHF/UHF:

- überwiegend Sichtverbindung
- örtlicher Funkverkehr
- gleichmäßigere Bedingungen

Umgangsformen

1. **Blockieren Sie den Empfänger nicht:** andere möchten auch hören
 2. **Nutzen Sie den Chat respektvoll:** seien Sie höflich zu anderen Nutzern
 3. **Melden Sie Probleme:** helfen Sie dem Betreiber, die Station zu pflegen
 4. **Fragen Sie nicht nach technischem Support:** dies ist eine Hörplattform. Schreiben Sie dem SysOp, wenn Sie Hilfe brauchen.
-

Fehlerbehebung

Kein Ton

Mögliche Ursachen:

1. Browser stummgeschaltet → Lautstärkeregelung des Browsers prüfen
2. System stummgeschaltet → Lautstärke des Rechners prüfen
3. Schwaches Signal → auf ein stärkeres Signal abstimmen (S7+)
4. Falsche Betriebsart → andere Demodulationsarten ausprobieren

Lösungen:

1. Auf ein starkes Signal klicken (orange/rot im Wasserfall)
2. Prüfen, ob der Browser nicht stummgeschaltet ist (Stummsymbol im Tab)
3. Eine andere Frequenz ausprobieren
4. Die Seite neu laden (F5)

Verzerrter Ton

Mögliche Ursachen:

1. Übersteuertes Signal → Signal zu stark
2. Falsche Betriebsart → AM-Signal in SSB gehört usw.
3. Störungen → Nachbarsignale strahlen ein

Lösungen:

1. Lautstärke verringern
2. Eine andere Demodulationsart ausprobieren
3. Ein schmaleres Filter verwenden
4. Von störenden Signalen wegstimmen

Ton bricht ab oder hinkt hinterher

Mögliche Ursachen:

1. Unregelmäßige Verbindung → Pakete kommen zu spät, und der Puffer läuft leer
2. Ausgelastetes Telefon oder ausgelasteter Computer → der Browser kommt nicht hinterher

Lösungen:

1. Erhöhen Sie den Puffer um eine Stufe (siehe *Audiopuffer und Verzögerung*)
2. Ziehen Sie WLAN oder Kabel mobilen Daten vor
3. Schließen Sie andere Tabs und Apps

Für einen genaueren Blick öffnen Sie die Seite mit angehängtem `?diag=1` (zum Beispiel `http://your_server:PORT/mobile?diag=1`). Die Seite `/mobile` zeigt dann unter ihren Bedienelementen die Verzögerung, wie unregelmäßig die Verbindung ist und wie es um den Puffer steht; die Desktop-Seite schreibt dieselbe Zeile alle zehn Sekunden in die Browserkonsole (F12). Nützlich, wenn Sie dem Betreiber der Station ein Problem melden.

Wasserfall wird nicht aktualisiert

Mögliche Ursachen:

1. Netzwerkproblem → langsame oder unterbrochene Verbindung

2. Browserleistung → zu viele geöffnete Tabs
3. Serverüberlastung → zu viele Nutzer

Lösungen:

1. Internetverbindung prüfen
2. Nicht benötigte Browser-Tabs schließen
3. Seite neu laden (F5)
4. Später erneut versuchen, wenn weniger Nutzer online sind

Abstimmen auf eine Frequenz nicht möglich**Mögliche Ursachen:**

1. Frequenz außerhalb des Bereichs → das SDR deckt diese Frequenz nicht ab
2. Falsches Eingabeformat → richtiges Format verwenden (z. B. „14200“ statt „14.200.000“)

Lösungen:

1. Frequenzabdeckung des SDR prüfen (auf der Seite angegeben)
2. Die angegebenen Formatbeispiele verwenden
3. Stattdessen in den Wasserfall klicken

Stotternder/abgehackter Ton**Mögliche Ursachen:**

1. langsame Internetverbindung
2. hohe Serverlast
3. Leistungsprobleme des Browsers

Lösungen:

1. Andere Anwendungen schließen, die Bandbreite verbrauchen
2. Außerhalb der Stoßzeiten erneut versuchen
3. Nicht benötigte Browser-Tabs schließen
4. Kabelverbindung statt WLAN verwenden

Häufig gestellte Fragen**Allgemeine Fragen**

F: Brauche ich besondere Ausrüstung, um WebSDR zu nutzen? A: Nein! Nur einen Computer oder ein Mobilgerät mit Internetzugang.

F: Ist die Nutzung von WebSDR kostenlos? A: Ja, die meisten WebSDRs sind kostenlos. Sie werden von Freiwilligen betrieben.

F: Kann ich mit WebSDR senden? A: Nein, WebSDR ist reiner Empfang. Senden ist nicht möglich.

F: Welche Frequenzen kann ich hören? A: Das hängt von der Konfiguration des WebSDR ab. Sehen Sie in den Stationsinformationen nach.

F: Kann ich den Ton aufzeichnen? A: Einige Browser erlauben Aufnahmen. Prüfen Sie die Funktionen Ihres Browsers.

Technische Fragen

F: Welche Abtastrate verwendet das SDR? A: Das ist von Station zu Station verschieden. Sehen Sie auf der Stationsinformationsseite nach.

F: Wie groß ist die Latenz? A: Meist einige Zehntelsekunden zwischen Funksignal und Ihren Lautsprechern: die Verarbeitung im Empfänger, das Netz und der Wiedergabepuffer des Browsers, der sich an Ihre Verbindung anpasst (etwa 100–150 ms auf einem Telefon im heimischen WLAN). Siehe *Audiopuffer und Verzögerung*.

F: Kann ich mehrere Instanzen nutzen? A: Meist ja, das kann den Server aber belasten. Bitte nehmen Sie Rücksicht.

F: Funktioniert es offline? A: Nein, WebSDR erfordert eine Internetverbindung.

F: Welche Browser werden unterstützt? A: Chrome, Firefox, Edge, Safari (jeweils aktuelle Versionen)

Fragen zur Nutzung

F: Wie viele Personen können gleichzeitig hören? A: Das hängt von der Serverkapazität ab. Oft 50-200 Nutzer und mehr.

F: Kann ich sehen, was andere hören? A: Falls aktiviert, ja. Achten Sie auf die Anzeigen für „andere Nutzer“.

F: Kann ich mit anderen Hörern chatten? A: Wenn der Betreiber ihn aktiviert hat. Suchen Sie das Chatfeld; ↪ **Reply** beantwortet eine bestimmte Nachricht (siehe *Chat* oben).

F: Warum ist auf manchen Frequenzen nichts zu sehen? A: Auf dieser Frequenz sind gerade keine Signale. Probieren Sie andere!

F: Was sind die farbigen Bänder im Wasserfall? A: Das Bandplan-Overlay mit den Frequenzzuweisungen.

Ressourcen

Mehr über Funk erfahren

- **Bandpläne:** suchen Sie nach „amateur radio band plan“ + Ihre Region
- **Ausbreitung:** informieren Sie sich über die Ausbreitung von KW-Funkwellen
- **Digitale Betriebsarten:** informieren Sie sich über FT8, PSK31, RTTY
- **Amateurfunk:** erwägen Sie, eine Amateurfunkgenehmigung zu erwerben!

Weitere WebSDRs finden

- **WebSDR-Verzeichnis:** <http://sdr-list.xyz>
- **WebSDR.org:** <http://websdr.org>
- **KiwiSDR:** <http://kiwisdr.com/public/>

Hilfe erhalten

1. **Stationsbetreiber:** Kontaktdaten auf der Seite prüfen
2. **Nutzerchat:** andere Hörer fragen (falls verfügbar)
3. **Onlineforen:** nach WebSDR-Gemeinschaften suchen
4. **Dokumentation:** dieses Handbuch zurate ziehen!

Anhang: übliche Frequenzen

KW-Amateurfunkbänder

Band	Frequenzbereich	Betriebsart	Aktivität
160 m	1.800-2.000 MHz	LSB	nachts/lokal
80 m	3.500-4.000 MHz	LSB	nachts/regional
40 m	7.000-7.300 MHz	LSB	tags/nachts/DX
30 m	10.100-10.150 MHz	USB	nur Daten/CW
20 m	14.000-14.350 MHz	USB	tagsüber/DX
17 m	18.068-18.168 MHz	USB	tagsüber/DX
15 m	21.000-21.450 MHz	USB	tagsüber/DX
12 m	24.890-24.990 MHz	USB	tagsüber/DX
10 m	28.000-29.700 MHz	USB	sporadisch/DX

VHF-/UHF-Amateurbänder

Band	Frequenzbereich	Betriebsart	Aktivität
6 m	50.000-54.000 MHz	USB/FM	sporadisch
2 m	144.000-148.000 MHz	FM	sehr rege
70 cm	420.000-450.000 MHz	FM	rege

Rundfunkbänder

Dienst	Frequenzbereich	Betriebsart
AM-Rundfunk	530-1710 kHz	AM
Kurzwelle	2.3-26.1 MHz	AM
UKW-Rundfunk	88-108 MHz	WBFM

Luftfahrt

Dienst	Frequenzbereich	Betriebsart
Flugsicherung	118-137 MHz	AM
ACARS (Daten)	130-136 MHz	Daten

Seefunk

Dienst	Frequenzbereich	Betriebsart
UKW-Seefunk	156-162 MHz	FM
KW-Seefunk	2-22 MHz	USB

Glossar

AGC: automatische Verstärkungsregelung – passt die Audiopegel automatisch an

AM: Amplitudenmodulation – Sprechfunkart für Luftfahrt und Rundfunk

Bandbreite: der Frequenzbereich eines Signals

CW: Continuous Wave – Morsesignale

DX: Weitverbindung

FFT: schnelle Fourier-Transformation – wandelt den Zeit- in den Frequenzbereich

FM: Frequenzmodulation – Sprechfunkart für VHF/UHF

HF: Kurzwelle (3-30 MHz) – Bänder für große Entfernungen

kHz: Kilohertz (1.000 Hz)

LSB: unteres Seitenband – Sprechfunkart für die unteren KW-Bänder

MHz: Megahertz (1.000.000 Hz)

NB: Störaustaster – entfernt Impulsstörungen

NR: Rauschminderung – vermindert Hintergrundrauschen

PSK: Phasenumtastung – digitale Betriebsart

RTTY: Funkfern schreiben – digitale Textbetriebsart

S-Meter: Signalstärkeanzeige

SDR: Software Defined Radio

SQL: Rauschsperrung – schaltet den Ton bei fehlendem Signal stumm

SSB: Einseitenband (USB oder LSB)

USB: oberes Seitenband – Sprechfunkart für die oberen KW-Bänder

VHF: Ultrakurzwelle (30-300 MHz) – Sichtverbindung

UHF: Dezimeterwelle (300-3000 MHz) – Sichtverbindung

Wasserfall: visuelle Darstellung des Funkspektrums über die Zeit

Viel Freude beim Erkunden des Funkspektrums mit PhantomSDR-Plus!

73 (beste Grüße) de SV1BTL & SV2AMK

Installationsanweisungen finden Sie in [INSTALLATION.md](#). Technische Einzelheiten finden Sie in [PROJECT_STRUCTURE.md](#).

KAPITEL 13

KiwiSDR-Client-Emulation

Aether_config.md

Emulation von KiwiSDR-Clients

AetherSDR, kiwiclient und andere KiwiSDR-Software mit PhantomSDR-Plus verbinden

Seit v4.1.0 kann PhantomSDR-Plus auch das **KiwiSDR-Protokoll** beantworten, sodass Software, die für einen KiwiSDR geschrieben wurde — **AetherSDR**, **kiwiclient** und die übrigen — sich direkt mit Ihrem Empfänger verbindet. Es ist eine Brücke innerhalb desselben Serverprozesses, auf demselben Host und demselben Port, den Sie ohnehin veröffentlichen: kein zweiter Dienst, kein zweiter Port, kein Proxy.

Sie ist aus, bis Sie sie einschalten. Nichts an der Brücke läuft — kein Socket, keine Prüfung — solange `[kiwi_emulation]` fehlt oder `false` ist, und das ist der Zustand eines frisch installierten Baums.

Inhaltsverzeichnis

1. Was ein Kiwi-Client bekommt
2. Die Brücke installieren
3. Einschalten
4. Konfigurationsreferenz
5. Einen Client verbinden
6. Lautstärke
7. Das S-Meter
8. Wasserfall und Spektrum
9. Wenn etwas nicht stimmt
10. Wie weit das geprüft wurde

1. Was ein Kiwi-Client bekommt

Audio	<code>ws://<host>:<port>/kiwi/<id>/SND</code> — demoduliertes Audio in der SND-Rahmung des Kiwi
Wasserfall	<code>ws://<host>:<port>/kiwi/<id>/W/F</code> — das Spektrum in Kiwi-W/F-Rahmen
Abstimmen	echt, nicht kosmetisch: <code>SET mod=...</code> vom Client ändert Frequenz, Seitenband und Durchlassbereich auf der PhantomSDR-Seite, genau wie es ein Browser-Hörer täte
S-Meter	gespeist aus der blockweisen Leistung des Demodulators, auf derselben Skala wie das Web-S-Meter

Das Abstimmen wird für einen **reellen** Eingang unterstützt (`signal = "real"` in Ihrer `.toml`). Bei einem IQ-Eingang ist die Beziehung zwischen Bin und Frequenz eine andere und nicht implementiert; ein IQ-Empfänger liefert also Audio und Wasserfall, folgt aber der Abstimmung eines Kiwi-Clients nicht.

2. Die Brücke installieren

Die Brücke besteht aus Patches für die Backend-Quellen plus einer neuen Header-Datei, `src/kiwi_bridge.h`. Es gibt zwei Wege, sie einzuspielen.

Mit dem Installationsprogramm. `install.sh` (sowie `install_arch.sh`, `install_fedora.sh`, `install_opensuse.sh`) haben einen eigenen Schritt dafür — Schritt 17, *Kiwi client emulation*. Er wird angeboten, nicht erzwungen: Antworten Sie `n`, und es wird nichts gepatcht. Für einen unbeaufsichtigten Lauf entscheidet `PHANTOM_KIWI=y|n`; die Vorgabe ist ja.

Auf einem bereits installierten Baum führen Sie das Skript aus, das im Wurzelverzeichnis mitgeliefert wird:

```
cd ~/PhantomSDR-Plus
./kiwi_install.sh
./recompile.sh          # [1] Nur Backend wählen
```

`kiwi_install.sh` sichert jede Datei, die es anfasst, zuerst nach `backup_kiwi_bridge_<zeitstempel>/` und wendet jeden Patch über einen exakten Textvergleich an — findet es einen Anker nicht dort, wo es ihn erwartet (lokal veränderter Baum, andere Version), hält es sofort an, nennt die Datei und lässt sie unangetastet. Es ist idempotent: Auf einem Baum, der die Brücke bereits hat, meldet es jeden Patch als bereits angewendet und ändert nichts.

Wenn Sie gerade ein Update-Archiv eingespielt haben, das die Brücke bereits enthält, brauchen Sie es nicht auszuführen. Die gepatchten Quellen liegen im Archiv; `kiwi_install.sh` würde Ihnen nur sagen, dass jeder Patch schon vorhanden ist.

Gepatcht werden `src/client.h`, `src/signal.cpp`, `src/waterfall.cpp`, `src/spectrumserver.h`, `src/spectrumserver.cpp`, `src/websocket.cpp` und `src/http.cpp`; `kiwi_bridge.h` wird nach `src/` kopiert.

Es installiert die Brücke, es aktualisiert sie nicht. Schlägt ein Patch auf einem Baum *fehl*, auf dem die Brücke bereits funktioniert, ist die Quelle dem Skript vorausgeeilt — dann muss das Skript nachgezogen werden, nicht Ihr Baum.

3. Einschalten

Fügen Sie dies der `.toml` hinzu, mit der Ihr Empfänger tatsächlich läuft — derselben Datei, die Sie an `spectrumserver` übergeben, der in Ihrem `start-<radio>.sh` genannten. Nicht `config.example.*`:

```
[kiwi_emulation]
enabled = true
```

Dann den Empfänger neu starten:

```
./stop-websdr.sh
./start-<ihr-radio>.sh
```

`kiwi_install.sh` fügt den Block mit `enabled = true` für Sie in diejenigen von `config.toml`, `config-rx88mk2.toml`, `config-airspyhf.toml`, `config-rtl.toml`, `config-rspla.toml`, `config-fobos.toml`, `config-fobos-hf.toml` und `config-hackrf.toml` ein, die im Wurzelverzeichnis vorhanden sind. Von einer Konfiguration, die Sie anderswo oder unter anderem Namen führen, kann es nichts wissen, und **Update-Archive liefern niemals eine `.toml` mit**, sodass der Block nach einem Update von Hand zu setzen ist. Die mitgelieferten `config.example.*.toml` enthalten ihn dokumentiert und auf `false`.

4. Konfigurationsreferenz

Alles, was die Brücke liest, steht in einem einzigen Abschnitt `[kiwi_emulation]`. Nur `enabled` ist erforderlich; die vier Feineinstellungen haben alle den kalibrierten Wert als Vorgabe, ein Block mit nichts als `enabled = true` ist also bereits richtig.

Schlüssel	Vorgabe	Wirkung
<code>enabled</code>	<code>false</code>	Das KiwiSDR-Protokoll überhaupt beantworten. Aus = vollständig untätig.
<code>audio_gain</code>	<code>0</code>	dB. Ausgangsverstärkung, nur für Kiwi-Clients. Siehe Lautstärke .
<code>smeter_offset</code>	<code>input.analog_smeter_offset</code>	dB. Überschreibt den S-Meter-Offset, nur für Kiwi-Clients. Siehe Das S-Meter .
<code>wf_cal</code>	<code>0</code>	dB. Anzeigekorrektur für Wasserfall und Spektrum. Siehe Wasserfall und Spektrum .
<code>wf_fps_max</code>	<code>23</code>	Wasserfallbilder pro Sekunde. Siehe Wasserfall und Spektrum .

Ein vollständig ausgefüllter Block mit den Werten, mit denen der Empfänger dieses Projekts läuft:

```
[kiwi_emulation]
enabled      = true
audio_gain   = 60.0    # dB, nur Kiwi-Clients. 0 = unverändert.
wf_cal       = -10.0   # dB, Geschmackssache. 0 = stimmt mit dem S-Meter überein.
wf_fps_max   = 28      # Bilder/s. 23 ist das Protokollmaximum.
# smeter_offset = 5.0  # dB. Vorgabe ist input.analog_smeter_offset.
```

Jeder dieser Schlüssel wird beim Start gelesen. Eine Änderung ist ein **Neustart**, keine Neuübersetzung — `./stop-websdr.sh`, dann Ihr Startskript. Zum Nachstellen dieser Werte müssen Sie nie neu kompilieren.

Ganzzahlige und dezimale Schreibweise werden gleichermaßen akzeptiert: `28` und `28.0` werden identisch gelesen. Nur ein wirklich fehlender Schlüssel fällt auf seine Vorgabe zurück.

5. Einen Client verbinden

Richten Sie den Client auf dieselbe Adresse und denselben Port, die Ihre Hörer benutzen — kein Pfad, kein Präfix. AetherSDR fragt nach Host und Port; `kiwiclient` nimmt `-s` und `-p`:

```
# kiwiclient, 30 Sekunden 7100 kHz LSB aufzeichnen
python3 kiwirecorder.py -s ihr.empfaenger.example -p 8073 -f 7100 -m lsb --tlimit=30
```

Der Client stimmt ab, der Wasserfall füllt sich, das S-Meter zeigt an. Passiert überhaupt nichts, ist die Brücke mit ziemlicher Sicherheit noch abgeschaltet — siehe [Abschnitt 3](#).

6. Lautstärke

Das Erste, was den meisten auffällt: Ein Kiwi-Client klingt dünner als die PhantomSDR-Webseite. Das liegt nicht daran, dass die Brücke etwas verlöre: Ein Puffer speist jeden Encoder, und dieselbe Frequenz mit

demselben Durchlassbereich, gemessen über den Audiopfad des Browsers und über `/SND`, liefert identische Samples. Die Webseite ist laut, weil `audio.js` den Klang im Browser neu aufbaut — Bassanhebung, Bandpass, Präsenz-anhebung, ein Kompressor mit Makeup-Gain und der Lautstärkeregler. Ein Kiwi-Client hat davon nichts und kann es auch nicht bekommen.

`audio_gain` schließt die Lücke, nur für Kiwi-Clients:

```
[kiwi_emulation]
enabled      = true
audio_gain = 55.0      # dB, nur Kiwi-Clients. 0 = unverändert.
```

Beginnen Sie bei etwa 55–60 dB. 55 dB ist kein Rechenergebnis: Dort wurden ein Kiwi-Client und die PhantomSDR-Webseite auf gleichem Pegel gemessen, an einem starken Ortssender auf 729 kHz. In der Praxis wird oft etwas mehr bevorzugt — dieser Empfänger blieb bei 60 — wählen Sie es also nach Gehör.

Kaputtmachen können Sie das Audio damit nicht. Ein **vorausschauender Spitzenbegrenzer** sitzt zwischen der Verstärkung und der 16-Bit-Begrenzung: Jedes Sample wird um 4 ms verzögert, während die Verstärkung aus Audio berechnet wird, das noch nicht gesendet wurde. Eine laute Passage trifft also auf eine Verstärkung, die bereits für sie heruntergegangen ist, und Spitzen falten sich, statt abgeflacht zu werden. Bei 60 und selbst 70 dB — weit jenseits des Spielraums der reinen Verstärkung — wurde nicht ein Sample von einer Viertelmillion beschnitten. Unterhalb der Schwelle des Begrenzers ist die Verstärkung exakt 1,0; ein Empfänger, der das Audio nicht hart fährt, bekommt die Samples also unangetastet.

Ein zu hoher Wert kostet Lautstärke, nicht Qualität. Oberhalb von etwa 60 dB wird die zusätzliche Verstärkung schlicht weggeregelt: Gemessen stieg der Pegel von 55 auf 60 um etwa 1 dB und bewegte sich von 60 auf 70 kaum noch. Darüber hinaus kaufen Sie Kompression statt Lautstärke.

Nichts davon berührt die Webseite, und nichts davon bewegt das S-Meter — das kommt aus der Leistung des Demodulators, nicht aus den Audiosamples. Lauter drehen kann das Meter also nicht zum Lügen bringen.

7. Das S-Meter

Das Kiwi-S-Meter wird nicht separat kalibriert: Es bildet nach, was Ihre Webseite anzeigt, Stufe für Stufe. Die blockweise Leistung des Demodulators ist für beide der Ausgangspunkt; die Seite wendet dann `input.analog_smeter_offset` an, spreizt das Ergebnis um etwa -130 dBm und addiert ihren eigenen Anzeige-Offset. Die Brücke tut dasselbe, sodass ein Kiwi-Client und die Seite dasselbe Signal mit derselben Stärke zeigen — an einem starken Ortssender überprüft, wo beide -37 dBm lasen.

Wenn — und nur wenn — Kiwi-Clients anders anzeigen sollen als Ihre Seite, geben Sie der Brücke einen eigenen Offset:

```
[kiwi_emulation]
enabled      = true
smeter_offset = 5.0      # ersetzt input.analog_smeter_offset, nur für Kiwi-Clients
```

Vergleichen Sie die beiden Anzeigen mit demselben Filter — oder gar nicht. Der Wert kommt aus der Leistung im Durchlassbereich, ein breiteres Filter sammelt also mehr Rauschen und zeigt mehr an: Auf einem ruhigen Band maß dieselbe Frequenz bei 9 kHz Bandbreite 5,6 dB stärker als bei 2,4 kHz. Ein Kiwi-Client mit 6-kHz-Filter gegen eine Webseite mit 2,4 kHz weicht um mehrere dB ab, wie auch immer beide kalibriert sind. Gleichen Sie zuerst Betriebsart und Bandbreite an.

Das ist die Skala der Seite, keine physikalische. Das Web-Meter spreizt seinen Bereich der Lesbarkeit halber, sodass 10 dB tatsächlicher Signaländerung als etwa 11 angezeigt werden. Sie nachzubilden heißt, dass Kiwi-Clients mit Ihrem Empfänger übereinstimmen und von einem echten KiwiSDR um einen mit der Signalstärke wachsenden Betrag abweichen. Das ist der richtige Kompromiss, wenn Ihre eigene Seite die Referenz ist, an der alle messen; soll die Brücke lieber physikalisch ehrlich bleiben, ist dies der Abschnitt, den Sie ändern.

8. Wasserfall und Spektrum

Die dB-Skala — `wf_cal`

Jeder Wasserfall-Bin wird vor dem Senden in echte dBm umgerechnet, auf dieselbe Skala, auf die das S-Meter kalibriert ist. Ein Träger zeigt daher **bei jedem Zoom denselben Pegel**, so wie es ein echter KiwiSDR liefert, und das Spektrum stimmt über denselben Durchlassbereich mit dem S-Meter der Webseite überein.

```
[kiwi_emulation]
enabled = true
wf_cal = 0.0          # dB, nur Kiwi-Clients
```

wf_cal ist keine Kalibrierung. 0 ist der Wert, bei dem das Spektrum mit Ihrem S-Meter übereinstimmt, und dabei sollten Sie es belassen, wenn die Zahlen etwas bedeuten sollen. Es existiert, weil es Geschmackssache ist — und eine Frage dessen, was Ihr Client mit dem Bereich anstellt —, wie *stark* ein Spektrum aussehen soll: Der Empfänger dieses Projekts läuft mit `-10`, schlicht weil das in AetherSDR richtig aussieht. Ein Wert bewegt Spektrum und Wasserfall gemeinsam: Es sind dieselben Bytes auf der Leitung, und das Kiwi-Protokoll kann sie nicht getrennt skalieren. Sollen sie sich unterscheiden, muss das aus den Anzeigeregeln des Clients kommen.

Zur Prüfung der Skala vergleichen Sie den über den Durchlassbereich summierten Wasserfall mit dem S-Meter auf einer **ruhigen** Frequenz. Auf einem Träger schließt ein SSB-Durchlassbereich den Träger selbst aus, und die beiden sind nicht vergleichbar.

Die Bildrate — `wf_fps_max`

Ein Kiwi-Client fordert den schnellsten Wasserfall an, den er haben kann (`SET wf_speed=4`), und richtet sein Scrollen — und seine Spektrumsmittelung — nach der Rate, die der Server zu liefern verspricht. Weichen die beiden voneinander ab, wirkt die Anzeige träge, obwohl nichts tatsächlich zu spät kommt.

Der Empfänger erzeugt $2 \times \text{sps} / \text{fft_size}$ Spektren pro Sekunde. Kiwi-Clients werden aus jedem einzelnen davon bedient und auf die angeforderte Rate ausgedünnt; die Webseite behält ihren eigenen, langsameren Takt und bleibt unberührt.

```
[kiwi_emulation]
enabled = true
wf_fps_max = 23      # Bilder pro Sekunde
```

23 ist das vom KiwiSDR-Protokoll selbst genannte Maximum und die sichere Vorgabe. **Es gewinnt immer der kleinere Wert von `wf_fps_max` und der Eigenrate des Empfängers**, ein Heraufsetzen bewirkt also nur dann etwas, wenn Ihre FFT schnell genug ist, um Bilder übrig zu haben. Beispiel für einen Empfänger mit 60 MS/s und 4194304-Punkt-FFT — $2 \times 60000000 / 4194304 = 28,6$ Bilder pro Sekunde:

<code>wf_fps_max</code>	geliefert
fehlt (Vorgabe 23)	23,0 fps
28	28,0 fps
40	28,6 fps — die Eigengrenze des Empfängers gewinnt

Auch SET `wf_speed` vom Client wird beachtet: 0 aus, 1 = 1 fps, 2 ein Viertel des Maximums, 3 die Hälfte, 4 das Maximum. Ein Client an einer dünnen Leitung kann also weniger anfordern.

Ob ein Heraufsetzen hilft, entscheidet der Client, nicht der Server. Ein Client, der jedes Bild beim Eintreffen zeichnet, gibt Ihnen ein flüssigeres Scrollen; ein Client, der sich auf die erwarteten 23 taktet, stellt die zusätzlichen Bilder schlicht in die Warteschlange, und die Verzögerung wächst. Probieren Sie es aus, und wenn der Wasserfall träger statt flüssiger wirkt, stellen Sie wieder auf 23.

Was die übrige Verzögerung bestimmt

Wer der Latenz nachgeht: Das meiste davon steckt nicht in der Brücke. Das Analysefenster ist `fft_size / sps` breit — 70 ms bei einer 4194304-Punkt-FFT und 60 MS/s — und ein Sample wartet zusätzlich bis zur Hälfte davon, bis der Vorschub voll ist. `fft_size` zu halbieren halbiert beides und verdoppelt die Bildrate, halbiert aber auch Ihre Frequenzauflösung, von der die Schmalband-Decoder (WSPR, FT8, CW) leben. Auf den meisten Empfängern ist das ein schlechter Tausch für einige zehn Millisekunden. Es ist eine Entscheidung in `[input]` und nichts, woran die Brücke etwas ändern könnte.

9. Wenn etwas nicht stimmt

Jeder Befehl, den ein Kiwi-Client sendet, wird nach `/tmp/kiwi_retune.log` geschrieben — die Abstimmwünsche, die Betriebsartwechsel, die Anmeldung und alles, was die Brücke nicht erkannt hat. Das kostet nichts Messbares und ist dafür gedacht, dauerhaft dort zu bleiben. Verhält sich ein Client sonderbar, zeigt diese Datei, was er tatsächlich verlangt hat — was meist die ganze Antwort ist.

Symptom	Nachsehen bei
Client verbindet sich und fliegt sofort raus	<code>[kiwi_emulation] enabled = true</code> fehlt in der Konfiguration, mit der der Server gestartet wurde
Kein Audio, kein Wasserfall, keine Logzeilen	das Backend wurde nach <code>kiwi_install.sh</code> nicht neu übersetzt — <code>./recompile.sh</code> , Option <code>[1]</code>
<code>kiwi_install.sh</code> bricht bei einem Patch ab	die Meldung nennt Datei und Anker; der Baum weicht von dem ab, was der Patch erwartet, und diese Datei blieb unangetastet
Audio vorhanden, aber dünn und leise	erwartet — <code>audio_gain</code> setzen, siehe Abschnitt 6
Wasserfall scrollt träge	<code>wf_fps_max</code> , siehe Abschnitt 8
Abstimmen im Client bewirkt nichts	ein IQ-Eingang (<code>signal = "iq"</code>); das Abstimmen ist nur für reelle Eingänge umgesetzt
Eine geänderte Einstellung wirkt nicht	diese Schlüssel werden beim Start gelesen — Empfänger neu starten

Die AGC liegt beim Client. `SET agc=` von einem Kiwi-Client wird angenommen und ignoriert — das Audio kommt mit der AGC von PhantomSDR an, und die AGC-Regler des Clients wirken auf das, was er empfängt.

10. Wie weit das geprüft wurde

Das Leitungsformat wurde gegen einen laufenden Empfänger geprüft und Zeile für Zeile gegen den Quelltext der beiden Clients gelesen, auf die es zielt — AetherSDRs `KiwiSdrProtocol.cpp` und `kiwiclients kiwi/client.py`. Rahmenaufbau und -größe, Sequenznummern, Bytereihenfolge, die S-Meter-Skala, das Abstimmen und der Wasserfall stimmen überein. Die Brücke wurde außerdem **mit der AetherSDR-Desktop-Anwendung selbst als funktionierend bestätigt**, durch den Betreuer, gegen diesen Empfänger.

An diesem Empfänger gemessen, mit einem Testclient an den laufenden Sockets: Das Abstimmen greift in 35–50 ms, das Audio läuft in Echtzeit ohne Drift, und der Wasserfall liefert die eingestellte Rate ohne Stocken und ohne verlorene Bilder.

Sollte auf Ihrer eigenen Installation doch etwas falsch aussehen, ist `/tmp/kiwi_retune.log` die erste Anlaufstelle — und eine Rückmeldung ist in jedem Fall willkommen.

Siehe auch: [Installationsanleitung](#) · [Projektstruktur](#) · [Benutzerhandbuch](#)

KAPITEL 14

Empfangsdiversität

RECEIVE_DIVERSITY.md

PhantomSDR-Plus — Empfangsdiversität (Receive Diversity)

Kombinieren Sie Ihren Empfänger mit einem **zweiten Empfänger an einem anderen Ort** und hören Sie jeweils denjenigen, der gerade das bessere Signal hat. Fällt der eine Standort in einen Schwund, hat der andere ihn meist nicht — und die Verbindung bleibt lesbar.

Der zweite Empfänger kann ein weiterer PhantomSDR-Plus, ein KiwiSDR, ein UberSDR oder ein WebSDR sein. Bei den ersten drei läuft alles im Browser: keine Serveränderungen, keine Konfigurationsdateien, nichts zu installieren und kein besonderer Zugang zum anderen Empfänger. Ein WebSDR ist die Ausnahme und braucht ein kleines Programm auf Ihrem eigenen Server — siehe [Einen WebSDR verwenden](#) weiter unten.

Was es tut und was nicht

Diversität bedeutet hier **Auswahl**: Sie hören zu jedem Zeitpunkt den einen oder den anderen Standort, mit einer kurzen Überblendung beim Wechsel. Es ist keine Phasengruppe und addiert die beiden Signale nicht.

Das ist bewusst so, und der Grund ist Physik, nicht Software. Zwei Empfänger in Hunderten von Kilometern Entfernung hören dieselbe Aussendung über unterschiedliche ionosphärische Wege. Die beiden Audio-Signalverläufe treffen mit unabhängiger Phase und leicht unterschiedlichem Doppler ein; addiert man sie, klingt das hohl und kammgefiltert — der klassische „phasige“ Effekt. Echtes kohärentes Zusammenführen braucht zwei Empfänger an einem **gemeinsamen Takt**, abtastgenau ausgerichtet; zwei unabhängige Empfänger über das Internet können das nie leisten.

Der Gewinn ist deshalb **Kontinuität, nicht Signalstärke**:

- Auf einem stabilen Weg, auf dem ein Empfänger schlicht besser ist, hören Sie diesen und gewinnen nichts. Das ist normal.
- Auf einem flatternden Weg mit tiefem QSB sind Schwunderscheinungen an zwei entfernten Standorten weitgehend unkorreliert. Standort A fällt drei Sekunden aus, Standort B nicht, und das Audio läuft weiter. Erwarten Sie *„ich verliere keine Wörter mehr“*, nicht *„aus S3 wurde S7“*.
- Der zweite Gewinn ist oft **lokales Rauschen und QRM**. Zwei Standorte haben unterschiedliche Nachbarn, unterschiedliche Stromnetze und unterschiedliches Splatter. Auf einem verrauschten Band kann das mehr ausmachen als der Schwund.

Es hat einen Preis: Sie hören **ein bis drei Sekunden hinter der Echtzeit**, weil der frühere Datenstrom verzögert werden muss, um zum späteren zu passen. Für Hören und Dekodieren unerheblich, störend beim Versuch, eine Station zu arbeiten.

HINWEIS

Die digitalen Decoder (FT8, JS8, WSPR, RADE und die übrigen) verwenden bewusst weiterhin Ihren **lokalen** Empfänger, nicht das kombinierte Audio. Diese Betriebsarten integrieren kohärent über einen ganzen Zeitschlitz, und ein Standortwechsel mitten im Schlitz ist eine Phasenunstetigkeit, die genau die Dekodierung kosten kann, die die Diversität retten sollte. Diversität bedient den Lautsprecher; die Decoder behalten den durchgehenden lokalen Datenstrom.

Schnelleinstieg

1. Öffnen Sie das Feld **Receive Diversity** — es sitzt direkt unter den Decoder-Fenstern, zugeklappt, mit der Anzeige `Receive Diversity – off`. Klicken Sie darauf, um es aufzuklappen.
2. Wählen Sie, was der zweite Empfänger ist: **PhantomSDR+**, **KiwiSDR**, **UberSDR** oder **WebSDR**.
3. Tragen Sie seine Adresse ein — oder wählen Sie eine gespeicherte — und drücken Sie **Start**.
4. Stimmen Sie Ihren eigenen Empfänger normal ab. Der zweite **folgt automatisch** — Frequenz, Betriebsart und Durchlassbereich — bei jeder Abstimmung.

Das ist die gesamte Einrichtung. Am anderen Empfänger ist nichts zu konfigurieren, und es gibt keine getrennte Abstimmung: er folgt Ihnen immer.

Adressen

Zweiter Empfänger	Was Sie eintragen	Hinweise
PhantomSDR+	host:8900	Der normale Port des Empfängers
KiwiSDR	host:8073	Der eigene Port des KiwiSDR
UberSDR	host	Sein normaler Web-Port , nicht 8073
WebSDR	host:8901	Der eigene Port der Station. Braucht das Relay — siehe unten

Ein reiner Hostname, eine `http://`-Adresse oder eine vollständige `ws://`-Adresse werden alle akzeptiert. Wird der andere Empfänger über HTTPS ausgeliefert, geben Sie den Hostnamen ein; die Verbindung wird dann verschlüsselt aufgebaut.

Gespeicherte Empfänger

Jede verwendete Adresse wird gemerkt, in einer eigenen Liste je Empfängertyp — eine KiwiSDR-Adresse ist nie ein brauchbarer Vorschlag, wenn die Auswahl auf WebSDR steht. Die Listen liegen in Ihrem eigenen Browser: sie werden nirgendwohin gesendet, und jeder Hörer hat seine eigene.

Die Schaltfläche  neben dem Adressfeld öffnet die Liste.

- **Namen.** Jeder Eintrag kann einen Namen bekommen — `Twente` liest sich besser als `websdr.ewi.utwente.nl:8901`. Benannte Einträge erscheinen grün über ihrer Adresse, und beim Tippen steht der Name neben der Adresse in der Vorschlagsliste. Ein Eintrag wird über seine Adresse identifiziert, ein Name bringt dieselbe Station also nie zweimal in die Liste. Solange die Diversität läuft, zeigt der gerade verwendete Empfänger seinen Namen **fett und langsam blinkend**, sodass die Liste auch sagt, welchen Sie tatsächlich hören.
- **add** trägt einen Empfänger von Hand ein, ohne ihn vorher zu verbinden.  korrigiert Name und Adresse an Ort und Stelle, **x** entfernt einen Eintrag, und **clear all** leert die Liste des gerade gewählten Typs. Enter speichert, Escape bricht ab.
- ******** öffnet die eigene Webseite dieses Empfängers in einem neuen Tab. Gespeichert ist die Adresse, die die Software auswählt, sie wird also zuerst in eine zum Browsen taugliche verwandelt — aus `ws://`

wird `http://`, und der Pfad `/audio`, `/ws` oder beim KiwiSDR `/kiwi/.../SND`, den die jeweilige Empfängerart braucht, fällt weg, denn die Seite der Station liegt an der Wurzel. Es ist ein echter Link, mittlere Maustaste und langes Drücken verhalten sich wie gewohnt.

- **Die Reihenfolge gehört Ihnen.** Verschieben Sie eine Zeile mit ▲▼, oder ziehen Sie sie an die Stelle, an die sie gehört. In dieser Reihenfolge bietet die Vorschlagsliste die Adressen an.
- **export und import.** Die Listen liegen im Speicher des Browsers, der beim Löschen der Websitedaten verschwindet — genau davor schützt dies. Export schreibt alle vier Listen samt Namen in eine JSON-Datei, die Sie aufheben oder auf einen anderen Browser oder Rechner mitnehmen können. Import liest sie zurück und fragt, ob sie mit dem Gespeicherten **zusammengeführt** (merge) oder ob alles **ersetzt** (replace) werden soll.
- **QR.** Zeichnet alle Listen als QR-Code auf den Bildschirm, um sie auf ein Telefon zu bringen — siehe „Auf dem Telefon“ weiter unten. Eine zum Scannen zu große Liste weicht auf den Text daneben aus, der sich überall einfügen lässt.

Wie viele Empfänger Sie behalten, ist nicht begrenzt.

Auf dem Telefon

Die Seite `/mobile` bietet dieselbe Funktion in einem Reiter **Div**, neben Audio, Bands, Marks, Users und Chat: Empfängertyp, Adresse, Start und Stop, die gespeicherten Empfänger als Liste zum Antippen, die Live-Werte und der SNR-Trim. Jede Zeile trägt außerdem den Link `****` zur eigenen Seite des Empfängers. Der zweite Empfänger folgt der Abstimmung des Telefons von selbst, genau wie auf der Desktop-Seite. Was fehlt, ist das Bearbeiten — Umbenennen, Umsortieren, Löschen —, das auf der Desktop-Seite bleibt, wo dafür Platz ist.

Die gespeicherte Liste gehört zu einem Browser, ein Telefon beginnt also mit einer leeren, wie viele Empfänger auch immer auf dem Rechner gespeichert sind. Dafür ist der QR-Code da: am Rechner ☰ und dann **QR** drücken, den Code mit der Kamera des Telefons scannen und den Text in **Import** im Reiter Div einfügen. Es fragt nach merge oder replace, und Export auf dem Telefon schickt eine Liste in die Gegenrichtung.

Einen WebSDR verwenden

Ein WebSDR — die Software von Pieter-Tjerk de Boer, PA3FWM, die in Twente und an mehreren hundert weiteren Standorten läuft — lässt sich als zweiter Empfänger nutzen, aber nicht direkt aus dem Browser.

Der Grund ist eine bewusste Prüfung auf deren Seite. Ein WebSDR verweigert die Audioverbindung, wenn der `Origin`-Header nicht seine eigene Seite nennt, und `Origin` ist ein *verbotener Headername*: der Browser setzt ihn aus der Seite, auf der Sie sind, und kein Skript darf ihn ändern. Auf der Clientseite gibt es keinen Weg daran vorbei — und es sollte auch keinen geben.

Die Verbindung stellt deshalb ein kleines Programm auf Ihrem eigenen Server her, `websdr_relay.py`. Ihr Browser spricht mit dem Relay, und das Relay spricht mit dem WebSDR.

Das Relay installieren

Die vier Distributions-Installer bieten es als optionalen Schritt an. Für eine bestehende Installation:

```
cd ~/PhantomSDR-Plus
./setup_websdr_relay.sh
```

Es fragt nach einem Port, übernimmt Rufzeichen und Adresse Ihrer Station aus `frontend/site_information.json` und bietet an, einen systemd-Dienst einzurichten, damit das Relay beim Booten startet. Die Einstellungen stehen in `websdr_relay.json` und lassen sich jederzeit ändern.

Eines kann der Installer nicht für Sie tun: den Port des Relays in Ihrem Router freigeben. Die Browser der Zuhörer verbinden sich direkt mit dem Relay — es läuft nicht über den Empfänger — ohne diese Freigabe funktioniert WebSDR-Diversity also nur aus Ihrem eigenen Netz. Die anderen drei Empfängertypen sind davon nicht betroffen.

Ist das Admin-Panel installiert, zeigt sein Dashboard eine Karte **WebSDR Diversity Relay** mit Zustand, Port und der Zahl der laufenden Sitzungen samt Gegenstellen.

Ein guter Gast sein

Das Relay verbindet sich mit fremden Empfängern von Ihrem Server aus statt von der Adresse jedes einzelnen Zuhörers, deshalb ist es auf gutes Benehmen ausgelegt:

- **Höchstens zehn gleichzeitige Sitzungen zu einem WebSDR**, sechzig insgesamt. Das verhindert, dass ein betriebsamer Tag bei Ihnen wie ein Angriff auf die Station eines anderen aussieht.
- **Ein User-Agent, der Ihre Station und deren Betreiber nennt**, damit ein Betreiber, der das nicht möchte, genau weiß, an wen er sich wenden kann. Bitte tragen Sie ihn ehrlich ein.
- **Nur Abstimmbefehle werden weitergereicht.** Über die Verbindung lässt sich nichts anderes senden.
- **Private, Loopback- und Carrier-NAT-Adressen werden abgelehnt**, damit niemand das Relay auf etwas in Ihrem Rechner oder Ihrem LAN richten kann.

WICHTIG

Die Origin-Prüfung gibt es, weil die WebSDR-Autoren ihre Empfänger nicht von fremden Seiten aus steuern lassen wollten. Das Relay zu benutzen ist eine bewusste Entscheidung, kein Versehen ihrerseits. Lassen Sie das Limit, wo es ist, halten Sie den User-Agent ehrlich, und hören Sie auf, wenn ein Betreiber Sie darum bittet.

Was bei einem WebSDR anders ist

- **Viele Stationen decken einige schmale Ausschnitte ab**, keinen durchgehenden Bereich — etwa ein 256-kHz-Fenster auf 40 m. Diversity greift nur innerhalb davon, und das Feld liest die Bandliste der Station, um zu wissen, wo diese liegen.
- **Die Audio-Abtastrate wechselt** mit der Filterbreite und liegt weiter von der Nennrate entfernt als bei den anderen Typen, das Ausrichten kann daher etwas länger dauern.
- **CW wird nicht angepasst** an die Konvention von WebSDR, die den Durchlassbereich vollständig unter den Träger legt. SSB und AM stimmen.

Das Feld lesen

Im Betrieb zeigt das Feld eine kleine Tabelle:

Zeile	Bedeutung
link	Die Verbindung zum zweiten Empfänger. ready ist das Ziel. Bei einem Fehler erscheinen der Fehlertext der Gegenstelle und der WebSocket-Schließcode
aligned	Die gemessene Verzögerung zwischen beiden Datenströmen nach dem Einrasten, sonst searching
corr	Wie stark die beiden Ströme korrelieren. Nur nach dem Ausrichten aussagekräftig
remote audio	Dekodiertes Audio vom zweiten Empfänger. none in Bernstein heißt: es wird nichts dekodiert
SNR local / remote	Die beiden verglichenen Signal-Rausch-Schätzungen
switches	Wie oft der Standort gewechselt wurde

Neben dem Titel sagt Ihnen ein Wort, woran Sie sind:

off (grau)	nicht aktiv
Please wait... (bernstein)	verbindet sich, oder verbunden und noch beim Ausrichten
Ready (grün)	verbunden, eingerastet und dem besseren Standort folgend
die Fehlermeldung der Gegenstelle (rot)	fehlgeschlagen, und es behebt sich nicht von selbst

Der Unterschied zwischen bernstein und rot ist der, der Zeit spart: bernstein heißt weiter warten, rot heißt aufhören zu warten und die Meldung lesen.

Der farbige Punkt sagt zusätzlich, welchen Standort Sie gerade hören — grün den eigenen Empfänger, cyan den entfernten.

Das Ausrichten dauert etwa 15 Sekunden

Das ist normal. Die beiden Ströme werden über die Korrelation ihrer **Audio-Hüllkurven** ausgerichtet, und dafür braucht es ein Zeitfenster. Die Suche läuft in zwei Stufen: eine schmale Suche über $\pm 1,5$ Sekunden kann beginnen, sobald etwa 9 Sekunden Ton gesammelt sind, und eine zweite, unabhängige Messung fünf Sekunden später muss mit der ersten **übereinstimmen**, bevor die Ausrichtung akzeptiert wird. Liegen die beiden Stationen zeitlich weiter auseinander — ein bis zwei Sekunden zusätzliche Pufferung irgendwo auf dem Weg —, übernimmt bei etwa 14 Sekunden die volle Suche über ± 4 Sekunden, und die Bestätigung folgt bei 19.

Dieser Bestätigungsschritt ist wichtig. Schwinden zwei Standorte gegenphasig — tragen also nie beide gleichzeitig das Signal —, rastet eine einzelne Korrelation bereitwillig auf eine überzeugte, völlig falsche Verzögerung ein, die wie ein Echo klingt. Zwei unabhängige, übereinstimmende Messungen zu verlangen, weist das zurück.

Eine zweite Sache muss stimmen, bevor eine Ausrichtung hält: die Abtastraten der beiden Empfänger. Zwei Empfänger sind zwei Takte und zwei Dezimierungsketten, ihre Audioströme treffen also **nicht** ganz mit derselben Rate ein, selbst wenn beide 12 kHz melden — 2% Unterschied sind normal, also 240 Abtastwerte pro Sekunde Drift, weit mehr als die Ausrichtungstoleranz. Der entfernte Strom wird deshalb laufend auf die Rate Ihres eigenen Empfängers resampelt. Dieses Verhältnis wird durch **Zählen von Abtastwerten** ermittelt, nicht durch Korrelation; es braucht also keine eigene Einrastung und steht schon in den ersten Sekunden fest — noch bevor die erste Ausrichtungssuche läuft. Ein schwaches oder unterbrochenes Signal kann zwei bis drei Anläufe brauchen, 30 Sekunden sind also kein Grund zur Sorge; eine Minute ohne Ergebnis schon.

Während `searching` zeigen beide SNR-Werte `0.0`. Das ist Absicht: Beide Schätzungen werden an **inhaltlich ausgerichteten** Abtastwerten gemessen, also wird vor einer Ausrichtung nichts gemessen. `0.0 / 0.0` heißt „noch nicht eingerastet“, nicht „kein Signal“.

Remote SNR trim

Dieser Regler gewichtet die Wahl zwischen den beiden Standorten. Er wird vor dem Vergleich zum gemessenen SNR des entfernten Empfängers addiert — positiv bevorzugt den entfernten, negativ den eigenen. Er ändert **sonst nichts**: weder den Pegel noch die Zusammenführung, nur wer gewinnt.

Er ist nötig, weil die beiden SNR-Werte nicht immer vergleichbar sind. Beide werden gleich gemessen — als Perzentilspanne des Audios —, aber ein Empfänger, dessen Codec sein eigenes Rauschen anhebt, liest niedriger als er verdient. Ein KiwiSDR arbeitet mit kräftiger Verstärkung und einem Limiter, was die Spanne staucht; ein verlustbehafteter Codec setzt einen Rauschteppich, unter den das Signal nie kommt.

So stellen Sie ihn ein:

1. Stimmen Sie beide Empfänger auf ein zuverlässig vorhandenes Signal ab und warten Sie auf `aligned`.
2. Beobachten Sie die beiden SNR-Werte eine halbe Minute lang. Merken Sie sich den üblichen Abstand.
3. Nutzen Sie die Endanschläge als Hörtest: **+15** erzwingt den entfernten Standort, **-15** den eigenen. Hören Sie jeden ein paar Sekunden. Nur so hören Sie einen Standort isoliert.
4. Liest der entfernte etwa 5 dB niedriger, klingt aber genauso gut, stellen Sie **+5** ein. Sie bringen die beiden Anzeigen zur Deckung, wenn beide Standorte gleich klingen.
5. Beobachten Sie den Zähler **switches** über zehn Minuten Hören. Ständiges Umschalten heißt, der Wert liegt zu nah am Gleichstand; nie umzuschalten, obwohl der eigene Empfänger hörbar schwindet, heißt, er ist zu weit in die andere Richtung verschoben.

Ausgangswerte: **0** für einen weiteren PhantomSDR-Plus, der denselben Audioweg nutzt und schon von der Konstruktion her vergleichbar ist; ein kleiner **positiver** Wert für einen KiwiSDR oder UberSDR.

TIPP

Brauchen Sie mehr als etwa ± 8 dB, hören Sie mit dem Nachstellen auf. Dann ist die ehrliche Erklärung meist, dass ein Empfänger für diesen Weg wirklich schlechter ist — und über einen echten Unterschied hinaus zu gewichten heißt nur, den schwächeren Standort zu hören.

Die Einstellung wirkt sofort und wird im Browser gespeichert.

Die Wahl des zweiten Empfängers

Entfernung zählt. Schwunderscheinungen entkorrelieren mit der Entfernung — einige hundert Kilometer auf Kurzwelle sind ein guter Zielwert. Zwei Empfänger in derselben Stadt schwinden gemeinsam und bringen nichts. Zu weit auseinander, und der zweite Empfänger hört Ihr Signal womöglich gar nicht.

Beide müssen das Signal tatsächlich hören. Diese Bedingung umgeht keine Software. Hört der zweite Empfänger nicht, was Sie hören, bleibt die Korrelation niedrig und er rastet nicht ein — mit Absicht, denn eine falsche Ausrichtung klingt schlechter als gar keine Diversität.

Abdeckung. Ein zweiter PhantomSDR-Plus oder ein KiwiSDR meldet den abgedeckten Frequenzbereich, und das Feld warnt Sie, wenn Sie außerhalb abstimmen. Ein WebSDR meldet seine Bänder, die oft

schmale Ausschnitte statt eines durchgehenden Bereichs sind. UberSDR meldet keine Abdeckung; dort ist das ausbleibende Einrasten Ihr einziger Hinweis.

WICHTIG

Öffentliche Empfänger werden von Freiwilligen betrieben und haben eine begrenzte Zahl an Hörerplätzen. Eine Diversitätssitzung belegt einen davon, solange sie läuft, genau wie ein menschlicher Zuhörer. Bitte lassen Sie eine Verbindung nicht unbegrenzt offen und fragen Sie den Betreiber, wenn Sie seinen Empfänger intensiv nutzen möchten.

Fehlersuche

Link verlässt connecting nie oder wechselt ständig zwischen Verbinden und Schließen

Wahrscheinlich stimmen Adresse oder Port nicht. Siehe Tabelle oben — vor allem nutzt UberSDR seinen normalen Web-Port, nicht 8073. Das Feld zeigt den WebSocket-Schließcode, die Browser-Konsole (F12) den zugrunde liegenden Fehler.

Link zeigt einen Fehler mit Text Dieser Text stammt vom anderen Empfänger und ist meist konkret: abgewiesene Sitzung, voller Server oder ein Empfänger, der keine fremden Clients annimmt.

Der KiwiSDR verweigert die Verbindung Manche KiwiSDRs verlangen ein Passwort, und auf einem UberSDR ist der KiwiSDR-kompatible Zugang standardmäßig abgeschaltet. Nutzen Sie für einen UberSDR die Quelle **UberSDR** — dieser Weg steht immer zur Verfügung.

WebSDR: the WebSDR relay is not reachable Das Relay läuft nicht, oder Ihr Browser erreicht seinen Port nicht. Es muss unter **demselben Hostnamen erreichbar sein, unter dem die Empfängerseite ausgeliefert wird**, denn der Browser verbindet sich direkt damit und nicht über den Empfänger. Öffnen Sie `http://<dieser Hostname>:<Relay-Port>/status` im selben Browser, um zu sehen, woran es liegt. Antwortet es im LAN, aber nicht von außen, ist der Port im Router nicht freigegeben.

WebSDR: ein Fehler über eine abgelehnte Verbindung Diese Station ist strenger eingestellt als die übliche Origin-Prüfung, oder ihr Betreiber hat diese Station gesperrt.

Link ist ready, aber remote audio zeigt none Es wird kein Audio dekodiert. Laden Sie die Seite neu; bleibt es dabei, zeigt die Browser-Konsole, welcher Decoder gescheitert ist.

Audio kommt an, richtet sich aber nie aus Die beiden Empfänger hören nicht dasselbe. Versuchen Sie einen starken Rundfunksender oder ein Band, auf dem beide Standorte gute Ausbreitung haben. Ein ruhiges Band mit beiderseits nur Rauschen korreliert nie — und soll es auch nicht.

Es schaltet ständig hin und her Die beiden Standorte liegen im SNR zu dicht beieinander. Verschieben Sie den Trimmwert leicht, sodass einer durchgehend bevorzugt wird.

Der zweite Empfänger wird nie verwendet Prüfen Sie, ob `remote audio` steigt und die SNR-Werte plausibel aussehen. Liest der entfernte deutlich niedriger, als er klingt, ist genau dafür der Trimmregler da.

Grenzen

- **Nicht kohärent.** Kein Phasing, kein Nullen, keine Peilung. Eine Störquelle auszulöschen braucht zwei Antennen an einem Takt an einem Standort; das ist ein anderes Verfahren für ein anderes Problem.
- **Ein bis drei Sekunden Verzögerung**, unvermeidlich.
- **Nur zwei Empfänger.**

- **Mono.** Läuft Ihr Empfänger in einer Stereo-Betriebsart wie C-QUAM, wird das Audio unverändert durchgereicht und die Diversität greift nicht.
- **Ausrichten braucht Signal.** Unter etwa 10 dB SNR an einem der beiden Standorte bleibt es erwartungsgemäß bei `searching`.
- **Ein WebSDR braucht das Relay** auf Ihrem eigenen Server und dessen freigegebenen Port. Die anderen drei Empfängertypen brauchen beides nicht.

KAPITEL 15

Transceiver-Steuerung

RIG_CONTROL.md

PhantomSDR-Plus — Transceiver-Steuerung (CAT)

Halten Sie **Ihren eigenen Transceiver** und einen **PhantomSDR-Plus-Empfänger** auf derselben Frequenz, Betriebsart und Filterbreite. Drehen Sie am Abstimmknopf des Funkgeräts, folgt der Wasserfall; klicken Sie auf ein Signal im Wasserfall, stimmt das Funkgerät darauf ab. Tasten Sie das Funkgerät, kann der Empfänger verstummen, damit er Ihnen nicht Ihr eigenes Signal zurückspielt.

Es funktioniert mit Empfängern **PhantomSDR-Plus, KiwiSDR, PA3FWM WebSDR und UberSDR**, Ihren eigenen oder fremden, und bewegt immer nur *Ihre* Hörsitzung — niemand sonst am Empfänger hört oder sieht etwas davon. Der Betreiber des Empfängers muss nichts installieren oder einstellen.

Was Sie brauchen

Zwei Wege, ein Funkgerät anzubinden, und eine Voraussetzung beim Empfänger:

Baustein	Was es ist	Synchronisiert
Desktop PhantomSDR+ ab 4.0	Die Desktop-Anwendung mit einem Menü Rig . Linux (PC und Raspberry Pi) und Windows.	Frequenz, Betriebsart, Filterbreite, Stummschaltung beim Senden — in eine Richtung oder in beide
CATsync Tool for WebSDRs	Ein eigenständiges Windows-Programm, das ein Funkgerät an die Empfängerseite im Browser koppelt.	Frequenz und Betriebsart
TCI-CAT auf der Empfängerseite	Eine Schaltfläche auf der PhantomSDR-Plus-Seite selbst, in jedem Browser. Spricht TCI mit ExpertSDR, AetherSDR oder Thetis, oder über eine kleine Brücke mit jedem Hamlib-Funkgerät. Nur auf PhantomSDR-Plus-Empfängern mit Version 4.1.0 oder neuer.	Frequenz, Betriebsart und Filterbreite in beide Richtungen; Stummschaltung beim Senden
Der Empfänger	PhantomSDR-Plus, KiwiSDR, PA3FWM WebSDR oder UberSDR. Ein PhantomSDR-Plus-Empfänger braucht für Filterbreite und Stummschaltung 4.1.0 oder neuer.	Ein älterer PhantomSDR-Plus synchronisiert weiterhin Frequenz und Betriebsart

Der größte Teil dieses Handbuchs beschreibt Desktop PhantomSDR+. Die Schaltfläche TCI-CAT hat einen eigenen Abschnitt, [TCI-CAT auf der Empfängerseite](#). Das CATsync Tool hat seine eigene Dokumentation auf seiner Website.

Unterstützte Empfänger

Die Anwendung erkennt, welche Art von Empfänger in einem Stationsfenster läuft, und steuert ihn über die eigenen Bedienelemente dieser Seite. Das Fenster Rig control zeigt die erkannte Art neben dem Namen der Station.

Empfänger	Frequenz und Betriebsart	Filterbreite	Stumm beim Senden
PhantomSDR-Plus	Ja	Mit 4.1.0 oder neuer	Mit 4.1.0 oder neuer
KiwiSDR (auch Web-888)	Ja	Ja	Ja
PA3FWM WebSDR	Ja, mit Bandwechsel auf Mehrband-Sites	Ja	Ja
UberSDR	Ja	Ja	Ja

Ein Web-Empfänger hat weniger Betriebsarten als die meisten Funkgeräte, daher teilen sich manche Betriebsarten des Geräts eine des Empfängers: KiwiSDR und WebSDR haben ein einziges CW für CW und CW-R. Die in diesem Handbuch genannten Empfänger-Betriebsarten sind die von PhantomSDR-Plus; die anderen Empfänger verwenden die jeweils nächstliegende. Ein WebSDR hält einen CW-Filter unter 1 kHz und andere Filter bei 1 kHz oder breiter, weil die Seite selbst CW daran erkennt, und eine Frequenz außerhalb aller Bänder einer WebSDR-Site wird nicht angefahren. UberSDR stellt seinen Filter in Schieberegler-Schritten ein, sodass eine Breite bis zu etwa 50 Hz von der des Geräts abweichen kann. Jede andere Art von Seite — etwa OpenWebRX — zeigt *not a receiver this app can drive*, und nichts wird synchronisiert.

Was es tut und was nicht

- Es synchronisiert **ein Funkgerät mit einem Empfängerfenster** zur selben Zeit.
- Es liest beide Seiten mehrmals pro Sekunde und stellt, wenn sie voneinander abweichen, die eine auf die andere ein. Es sendet **nicht**, tastet das Funkgerät nicht und schickt keinen Ton irgendwohin.
- Das Verstellen des Empfängers ändert nur Ihre eigene Sitzung. Andere Hörer am selben Empfänger merken nichts, und der Betreiber muss es nicht erlauben.
- Eine serielle Schnittstelle kann jeweils nur **ein Programm** öffnen. Hat WSJT-X, ein Logbuch oder ein Hersteller-Tool die Schnittstelle schon belegt, teilen Sie das Funkgerät über **flrig** oder **rigctlid im Netzwerk**, statt um die Schnittstelle zu streiten.

Schnellstart

1. Öffnen Sie wie gewohnt eine Station in Desktop PhantomSDR+.
2. **Rig → Rig control...**
3. Wählen Sie unter **Connection Built-in**, wenn Ihr Funkgerät in der Liste steht, sonst **Hamlib (all rigs)**.
4. Wählen Sie Ihr Funkgerät, die serielle Schnittstelle und die Geschwindigkeit, auf die das CAT- oder CI-V-Menü des Funkgeräts eingestellt ist.
5. Lassen Sie unter **Sync Both directions** ausgewählt.
6. Drücken Sie **Connect**. Die beiden Anzeigen oben — Transceiver und Empfänger — sollten innerhalb einer Sekunde dieselbe Frequenz zeigen.

Jede Einstellung wird gespeichert, sobald Sie sie ändern. Beim nächsten Mal genügt **Rig → Connect**, oder setzen Sie das Häkchen bei **Connect when the app starts**.

Den Verbindungsweg wählen

Auswahl	Verwenden, wenn	Benötigt
Built-in	Ihr Funkgerät in der Liste unten steht.	Sonst nichts
Hamlib (all rigs)	Ihr Funkgerät etwas anderes ist — Hamlib kennt mehr als 300 Geräte. Die Anwendung startet Hamlibs <code>rigctld</code> für Sie auf einem privaten lokalen Port und beendet ihn beim Trennen.	Windows: nichts, Hamlib ist enthalten. Linux: <code>sudo apt install libhamlib-utils</code>
rigctld on network	Bereits ein <code>rigctld</code> läuft, auf diesem oder einem anderen Rechner im Netzwerk.	Host und Port (Standard 4532)
flrig	flrig das Funkgerät bereits für fldigi, WSJT-X oder ein Logbuch steuert.	Laufendes flrig mit seinem XML-RPC-Port (Standard 12345)

Funkgeräte mit integriertem Treiber

Geschwindigkeit und CI-V-Adresse sind die Werkseinstellungen, die die Anwendung vorbelegt. **Sie sind nur ein Ausgangspunkt — stellen Sie ein, was im Menü Ihres Funkgeräts steht.**

Familie	Geräte	Standardgeschwindigkeit	Hinweise
Icom CI-V	IC-7300, IC-7610, IC-705, IC-9700, IC-905, IC-7760, IC-7851, IC-7100, IC-7410, IC-9100, IC-7600, IC-7200, IC-7700, IC-7000, IC-7800, IC-756PROIII, IC-756PROII, IC-R8600 und jedes andere CI-V-Gerät	19200	CI-V-Adresse je Modell vorbelegt (IC-7300 94, IC-705 A4, IC-9700 A2, IC-7610 98 ...)
	IC-746PRO, IC-718, IC-R75	9600	
Xiegu (CI-V)	G90, X6100	19200	Adresse 70; im Gerätemenü prüfen
Yaesu neues CAT	FTDX101D/MP, FTDX10, FT-710, FT-991/A, FT-891, FTDX5000, FTDX3000, FTDX1200, FT-950, FT-2000, FT-450/450D	38400	
Yaesu klassisches CAT	FT-817/818, FT-857/857D, FT-897/897D	38400	2 Stoppbits; stimmt in 10-Hz-Schritten ab
Kenwood	TS-990S, TS-890S, TS-590S/SG	115200	
	TS-480, TS-2000, TS-870S	57600	
Elecraft	K4, K3/K3S, KX3, KX2	38400	Filterbreite wird synchronisiert
Kenwood-kompatibel	FlexRadio SmartSDR CAT (virtueller Port), QRP Labs QMX/QMX+/QDX, (tr)uSDX, Lab599 Discovery TX-500, andere Kenwood-kompatible Geräte	9600–38400	

Ein Gerät, das kompatibel sein müsste, aber mit einem integrierten Treiber nicht spricht, funktioniert meist mit **Hamlib**, das mit weit mehr Varianten zurechtkommt.

Serielle Einstellungen

Einstellung	Was dort hineingehört
Serial port	Die Schnittstelle des Funkgeräts. USB-Adapter und Geräte mit USB-Anschluss stehen oben. Other / network address... nimmt eine Schnittstelle an, die nicht in der Liste steht — COM7, /dev/ttyUSB1 — oder tcp://host:port für eine serielle Schnittstelle, die per ser2net o. Ä. über das Netzwerk bereitgestellt wird.
Speed (baud)	Genau das, was im CAT-/CI-V-Baudraten-Menü des Funkgeräts steht. Eine falsche Rate sieht aus wie ein Gerät, das nie antwortet.
Stop bits	1 für fast alles; 2 für die Familie FT-817/857/897.
CI-V address	Nur Icom, hexadezimal (94, nicht 148). Muss zum CI-V-Adressmenü des Geräts passen.
DTR / RTS	Aus lassen, sofern Ihr Interface sie nicht braucht. Viele CAT-Kabel tasten auf einer dieser Leitungen den Sender oder setzen das Gerät zurück.
Hardware flow control	Aus lassen, außer das Handbuch verlangt RTS/CTS.

Bei **Hamlib** werden dieselben Einstellungen an `rigctl` weitergegeben. Stoppbits haben dort zusätzlich *Rig default*, und **Extra rigctl options** nimmt alles, was `rigctl` sonst versteht, z. B. `--set-conf=post_write_delay=10`. Mit **rigctl program** zeigen Sie auf einen bestimmten `rigctl`, wenn mehrere installiert sind.

Synchronisierung

Richtung

Auswahl	Was geschieht
Rig → receiver	Das Empfängerfenster folgt dem Funkgerät. Eine Änderung im Wasserfall wird auf die Frequenz des Funkgeräts zurückgesetzt.
Receiver → rig	Das Funkgerät folgt dem Empfängerfenster. Drehen am Abstimmknopf wird rückgängig gemacht.
Both directions	Die Seite, die Sie zuletzt berührt haben, gewinnt. Beim Verbinden, bevor eine Seite berührt wurde, gewinnt das Funkgerät.

Die Richtung lässt sich während der Verbindung auch im Menü **Rig** ändern.

Wie verhindert wird, dass sich beide Seiten bekämpfen

Jeder Wert, den die Anwendung schreibt, erscheint kurz darauf als Änderung auf der anderen Seite. Würde man das wörtlich nehmen, würden sich Funkgerät und Empfänger endlos gegenseitig jagen. Die Anwendung verhindert das auf drei Arten:

- Die Empfängerseite übernimmt eine Änderung sofort; sie wird direkt nach dem Schreiben zurückgelesen, und dieser Wert wird der neue Ausgangspunkt.
- Ein Funkgerät übernimmt eine Änderung etwas später; deshalb merkt sich die Anwendung jeden gesendeten Wert. Meldet das Gerät diesen Wert, wird er als eigener Schreibvorgang erkannt und nicht als Hand am Knopf.

- Ein Wert, den das Gerät ablehnt — etwa Breitband-FM auf einem KW-Gerät —, wird **zweimal** gesendet und dann in Ruhe gelassen, bis sich die Quellseite ändert, statt mehrmals pro Sekunde wiederholt zu werden.

Das Umstimmen der Empfängerseite kann dazu führen, dass sie die Standard-Betriebsart des Bandes wählt (unter 10 MHz etwa LSB). Führt das Funkgerät, setzt die Anwendung sofort wieder dessen Betriebsart, sodass ein Gerät in USB auf 40 m den Empfänger in USB hält.

Welches Empfängerfenster

Receiver window bestimmt, welche Station dem Funkgerät folgt:

- **The station window last in front** (Standard) — mit zwei offenen Stationen klicken Sie in eine hinein, und das Funkgerät folgt dieser.
- **Eine bestimmte Station** — das Funkgerät bleibt an ihr, ob sie vorne ist oder nicht. Ist diese Station nicht geöffnet, wird nichts synchronisiert, bis sie es ist.

Aktualisierungsrate

Update every legt fest, wie oft beide Seiten gelesen werden: 150 ms, 300 ms (Standard), 500 ms oder 1 s. Schneller fühlt sich am Knopf unmittelbarer an; langsamer schon ein altes Gerät mit 4800 oder 9600 Baud, bei dem jede Abfrage echte Zeit auf der Leitung kostet.

Betriebsarten

Betriebsart am Funkgerät	Empfänger hört in
USB, LSB	USB, LSB
CW	CW
CW-R (umgekehrt)	CW-L
AM, Synchron-AM, DSB	AM
FM, Schmal-FM	FM
Breitband-FM	WBFM
RTTY / FSK	LSB
RTTY-R / FSK-R	USB
Datenbetriebsarten (USB-D, DATA-U, PKTUSB, DIG)	USB
Daten-LSB, Daten-FM	LSB, FM

Betriebsart am Empfänger	Funkgerät wird gesetzt auf
USB, LSB	USB, LSB
CW	CW
CW-L	CW-R
AM, QUAM	AM
FM	FM
WBFM	WFM — die meisten KW-Geräte lehnen es ab und werden nach zwei Versuchen in Ruhe gelassen
RADE (oberes / unteres)	USB / LSB

Ein Gerät in einer **Datenbetriebsart** bleibt darin: USB am Empfänger gilt als übereinstimmend mit USB-D am Gerät, sodass der Empfänger das Gerät nie aus dem Datenbetrieb wirft.

Filterbreite

Setzen Sie das Häkchen bei **Sync filter width**, damit die Durchlassbereiche übereinstimmen. Unterschiede unter 60 Hz gelten als gleich, da keine zwei Filter gleich gestuft sind.

Verbindung zum Gerät	Filterbreite
Hamlib	Ja, soweit Hamlib es für dieses Gerät unterstützt
flrig	Ja
Integriert Icom CI-V	Ja — 50-Hz-Schritte bis 500 Hz, dann 100-Hz-Schritte bis 3,6 kHz; AM in 200-Hz-Schritten bis 10 kHz; nicht in FM
Integriert Elecraft	Ja, in 10-Hz-Schritten
Integriert Kenwood, Yaesu, FT-817-Familie	Nein — diese Geräte wählen Filter aus modellspezifischen Tabellen. Für den Filter Hamlib verwenden

KiwiSDR-, WebSDR- und UberSDR-Empfänger haben die Filtersteuerung immer. Ein PhantomSDR-Plus-Empfänger braucht **4.1.0** oder neuer; bei einem älteren werden Frequenz und Betriebsart weiterhin synchronisiert, und das Fenster Rig control sagt, warum der Filter nicht folgt.

Stummschaltung beim Senden

Setzen Sie das Häkchen bei **Mute receiver while transmitting**. Solange das Funkgerät getastet ist, ist das Empfängerfenster stumm; beim Loslassen kommt der Ton zurück. Die Stummschalttaste des Empfängers zeigt es an, und Sie können von Hand wieder einschalten.

Hatten Sie den Empfänger schon selbst stummgeschaltet, bleibt er auch danach stumm.

Es braucht eine Geräteverbindung, die den Sendezustand meldet — alle integrierten Treiber, flrig und Hamlib bei den meisten Geräten — und bei einem PhantomSDR-Plus-Empfänger 4.1.0 oder neuer.

Frequenzversatz

Frequency offset wird zur Frequenz des Funkgeräts addiert, um die des Empfängers zu erhalten:

Empfängerfrequenz = Gerätefrequenz + Versatz

Aufbau	Versatz
2-m-Transverter an einem 10-m-Gerät (144,100 MHz erscheint als 28,100 MHz)	116000000
70-cm-Transverter an einem 2-m-Gerät (432 → 144)	288000000
Kein Transverter	0

Das Menü Rig

Eintrag	Wirkung
Rig control...	Öffnet das Fenster Rig control
Connect / Disconnect <i>Gerätename</i>	Startet oder beendet die Synchronisierung; bei Hamlib auch rigctld
Rig to receiver / Receiver to rig / Both directions	Richtung der Synchronisierung
Sync filter width	Ein / aus
Mute receiver while transmitting	Ein / aus
Statuszeile	<i>Not connected, Connecting..., Connected: Gerätename</i> oder der letzte Fehler

Die laufende Frequenzanzeige steht im Fenster Rig control und nicht im Menü, das sich sonst bei jeder Änderung selbst schließen würde.

Linux

Rechte für die serielle Schnittstelle. Serielle Schnittstellen gehören der Gruppe `dialout`. Ein Benutzer außerhalb davon erhält *Could not open ttyUSB0*. Fügen Sie sich einmal hinzu und melden Sie sich dann ab und wieder an:

```
sudo usermod -aG dialout $USER
```

Hamlib. Aus der Distribution installieren:

```
sudo apt install libhamlib-utils
```

Das `.deb`-Paket von Desktop PhantomSDR+ empfiehlt es, daher bringt `sudo apt install ./phantomsdr-plus-desktop_4.0.0_amd64.deb` es mit; `dpkg -i` installiert empfohlene Pakete nicht. Die integrierten Treiber und flrig brauchen kein Hamlib.

Windows

Hamlibs eigenes `rigctl.exe` ist im 64-Bit- und im 32-Bit-Installer enthalten. COM-Schnittstellen erscheinen in der Liste mit ihrem Namen (`COM3`). Braucht das Gerät einen USB-Treiber, installieren Sie zuerst den des Herstellers — vorher existiert die Schnittstelle nicht.

TCI-CAT auf der Empfängerseite

Eine PhantomSDR-Plus-Empfängerseite kann einem Transceiver auch **selbst** folgen, ohne Desktop-Anwendung: Der Browser spricht direkt mit einem **TCI**-Server auf Ihrem eigenen Computer. TCI ist das WebSocket-Steuerprotokoll von ExpertSDR2/ExpertSDR3 (SunSDR), AetherSDR (FlexRadio) und Thetis (Apache Labs ANAN, Hermes). Für ein Funkgerät ohne TCI lässt eine kleine, in PhantomSDR-Plus enthaltene Brücke jedes Hamlib-Funkgerät wie einen TCI-Server aussehen.

Voraussetzung ist ein PhantomSDR-Plus-Empfänger mit **4.1.0 oder neuer**. KiwiSDR-, WebSDR- und UberSDR-Seiten haben diese Funktion nicht.

Die Schaltfläche TCI-CAT

Die Schaltfläche liegt in der Reihe mit **VFO**, **Modes**, **Bands** und **IF Filters** — **TCI-CAT** im breiten Layout, **CAT** im kompakten. Ihr Punkt ist **grün**, solange ein TCI-Server verbunden ist, sonst **grau**. Sie öffnet ein Fenster mit: Bis 4.2.0 hieß die Schaltfläche **QRG Sync**.

Element	Funktion
Status	<i>Active</i> — TCI (port 50001) bei bestehender Verbindung; <i>Inactive</i> mit <i>Wait</i> , solange gesucht wird
CAT Sync	Schaltet die Synchronisation von Frequenz, Betriebsart und Filter ein oder aus. Die Stummschaltung beim Senden wirkt in beiden Fällen
Host	Der Computer mit dem TCI-Server: <code>localhost</code> für diesen Computer, sonst seine Adresse im lokalen Netz, z. B. <code>192.168.1.42</code> . Mit Enter oder einem Klick daneben wird neu verbunden

Mehr ist nicht zu wählen: Die Ports **50001** (ExpertSDR3, AetherSDR) und **40001** (ExpertSDR2, Thetis) werden gleichzeitig probiert und alle paar Sekunden wiederholt, sodass die Seite sich verbindet, sobald die Software läuft. **CAT Sync** und **Host** merkt sich der Browser.

Browser, die fragen, bevor eine Webseite das lokale Netz erreichen darf — aktuelle Versionen von Chrome und Edge tun das — fragen einmal, beim ersten Verbindungsversuch. Erlauben Sie es, sonst bleibt der Punkt grau.

Was synchron gehalten wird

	Verhalten
Funkgerät → Empfänger: Frequenz	Nur die Skala bewegt sich. Zoom, Helligkeit und Kontrast bleiben, wie Sie sie eingestellt haben; das Wasserfalldiagramm scrollt nur, wenn die Frequenz außerhalb des Bildes liegt, und der Bandplan ändert die Betriebsart nicht
Funkgerät → Empfänger: Betriebsart	Folgt, wenn sich die Betriebsart des Funkgeräts ändert — USB, LSB, CW, AM, FM; Datenbetriebsarten als USB oder LSB. Eine Betriebsart, die der Empfänger nicht hat, lässt ihn unverändert. Ein laufender Decoder behält seine eigene Betriebsart
Funkgerät → Empfänger: Filterbreite	Der Durchlassbereich des Empfängers übernimmt die Filterbreite des Funkgeräts: Das Verstellen des Filters oder die Taste FIL am Funkgerät ändert den Durchlassbereich. Ein laufender Decoder behält seinen eigenen Durchlassbereich
Empfänger → Funkgerät: Frequenz	Eine eingetippte Frequenz, ein Label oder Lesezeichen, ein Klick in den Wasserfall und das Ziehen des Durchlassbereichs bewegen den VFO des Funkgeräts. Beim Ziehen werden nur geänderte Frequenzen gesendet
Empfänger → Funkgerät: Betriebsart	Jede Änderung der Betriebsart auf der Seite — eine Betriebsart-Taste, der Bandplan, ein Decoder, ein Lesezeichen — stellt die Betriebsart des Funkgeräts ein (USB, LSB, CW, AM, FM; RADE als USB oder LSB). Ein Funkgerät in einer Datenbetriebsart (USB-D) bleibt darin
Empfänger → Funkgerät: Filterbreite	Die Tasten unter IF Filters , der IF-Schieberegler, das Ziehen des Durchlassbereichs und ein Betriebsartwechsel stellen den Filter des Funkgeräts ein; beim Ziehen wird nur die endgültige Breite gesendet. Bei Icom-Geräten wählt die Brücke stattdessen FIL1, FIL2 oder FIL3 — siehe Icom-Filter
Kein Echo	Eine Änderung, die vom Funkgerät kam, wird nie an das Funkgerät zurückgesendet, und eine Meldung der vorherigen Betriebsart oder des vorherigen Filters, die kurz nach einer Änderung auf der Seite eintrifft, wird ignoriert
Stummschaltung beim Senden	Immer aktiv, auch mit CAT Sync aus: Die Seite verstummt, solange das Funkgerät sendet. Wird der Lautstärkeregler während des Sendens bewegt, bleibt sie stumm; beim Empfang kehrt der Ton mit der aktuellen Reglerstellung zurück
Nicht unterstützt	Split, VFO B, RIT/XIT und Transverter-Versatz. Nur VFO A des ersten Empfängers wird verfolgt

Betriebsart und Filterbreite werden mit den TCI-eigenen Befehlen (`modulation`, `rx_filter_band`) übertragen, daher sollten ExpertSDR, AetherSDR und Thetis ihnen ebenfalls folgen; ausprobiert wurde nur die Hamlib-Brücke. Für einen Transverter-Versatz verwenden Sie stattdessen Desktop PhantomSDR+. Benutzen Sie nicht beides gleichzeitig mit demselben Funkgerät — zwei Steuerungen kämpfen um die Skala.

Mit welchen Transceivern es funktioniert

Transceiver	Funktioniert	Wie
SunSDR (ExpertSDR2/3), FlexRadio (AetherSDR), ANAN/Hermes (Thetis)	Ja	Direkt — den TCI-Server in der Software aktivieren. Mit diesen Programmen noch nicht ausprobiert; getestet gegen einen simulierten TCI-Server
Ein Funkgerät mit CAT-Anschluss, das Hamlib unterstützt — die meisten von Icom, Yaesu, Kenwood, Elecraft, Xiegu, QRP Labs	Ja	Über die Hamlib-Brücke unten. Ausprobiert mit einem Icom IC-7300 unter Linux und Windows; andere Treiber können sich bei Betriebsartnamen oder der PTT-Meldung unterscheiden
Ein Funkgerät ohne CAT-Anschluss oder eines, das Hamlib nicht unterstützt	Nein	Es gibt nichts, woraus die Frequenz gelesen werden kann

Die Hamlib-Brücke (IC-7300 und andere Funkgeräte ohne TCI)

`tci-bridge/tci-rigctld.mjs` im PhantomSDR-Plus-Verzeichnis macht aus jedem Funkgerät, das Hamlib steuern kann, einen TCI-Server für die Seite. Sie läuft auf **Ihrem** Computer — dem, der mit dem Funkgerät verbunden ist, neben dem Browser — nicht auf dem Empfänger. Vermal pro Sekunde fragt sie Hamlib nach Frequenz, Betriebsart, Filterbreite und Sendezustand, schickt der Seite nur Änderungen und gibt die Frequenz-, Betriebsart- und Filteränderungen der Seite an das Funkgerät weiter.

Sie erreicht Hamlib auf einem von zwei Wegen:

Weg	Kette	Wann
--rigctl (empfohlen)	Transceiver → rigctl → tci-rigctld.mjs → Seite	Im Normalfall. Die Brücke startet Hamlibs <code>rigctl</code> selbst: ein Fenster, kein Netzwerkport. Unter Windows diesen Weg nehmen , wo <code>rigctld.exe</code> oft mit <i>Access is denied</i> abgewiesen wird
rigctld	Transceiver → rigctld → tci-rigctld.mjs → Seite	Ein anderes Programm, etwa WSJT-X, soll das Funkgerät gleichzeitig benutzen — siehe Das Funkgerät teilen

Was sie braucht

	Linux	Windows
Hamlib	<code>sudo apt install libhamlib-utils</code> oder das hamlib-Paket Ihrer Distribution	<code>hamlib-w64-....zip</code> von github.com/Hamlib/Hamlib/releases , entpackt nach <code>C:\hamlib</code> — <code>rigctl.exe</code> liegt in <code>C:\hamlib\bin</code>
Node.js	18 oder neuer	Der LTS-Windows Installer (.msi) von nodejs.org , mit den Standardoptionen
Das Paket <code>ws</code>	Wird im PhantomSDR-Plus-Verzeichnis automatisch gefunden	<code>tci-rigctld.mjs</code> in einen Ordner wie <code>C:\tci-bridge</code> kopieren und dort einmal <code>npm install ws</code> ausführen
Der Port des Funkgeräts	<code>ls /dev/serial/by-id/</code> . Einmal der Gruppe <code>dialout</code> beitreten: <code>sudo usermod -aG dialout \$USER</code> , dann ab- und wieder anmelden	Den USB-Treiber des Herstellers installieren; die COM-Nummer steht im Geräte-Manager → Anschlüsse (COM & LPT)

Nur ein Programm kann den CAT-Anschluss des Funkgeräts belegen. Schließen Sie WSJT-X, flrig, JS8Call, RS-B41 oder die Funkgeräteverbindung von Desktop PhantomSDR+, bevor Sie die Brücke starten.

Beispiel: Icom IC-7300

Am Funkgerät: **MENU** → **SET** → **Connectors** → **CI-V** — **CI-V USB Baud Rate** `115200`, **CI-V Transceive** ON. Hamlibs Modellnummer für den IC-7300 ist `3073`.

Linux. Das Funkgerät ist die Zeile mit `IC-7300` in `ls /dev/serial/by-id/`, meist auch `/dev/ttyUSB0`.

1. Prüfen, ob Hamlib das Funkgerät erreicht — es muss die Frequenz ausgeben, z. B. `14280000`: `bash rigctl -m 3073 -r /dev/ttyUSB0 -s 115200 f`
2. Die Brücke starten und laufen lassen: `bash cd ~/PhantomSDR-Plus/tci-bridge node tci-rigctld.mjs --rigctl rigctl -m 3073 -r /dev/ttyUSB0 -s 115200`

Windows. Das Funkgerät erscheint im Geräte-Manager als *Silicon Labs CP210x USB to UART Bridge (COM4)* — verwenden Sie Ihre eigene COM-Nummer. In einer Eingabeaufforderung:

1. Prüfen, ob Hamlib das Funkgerät erreicht — es muss die Frequenz ausgeben: `bat C:\hamlib\bin\rigctl.exe -m 3073 -r COM4 -s 115200 f`

2. Die Brücke starten und das Fenster offen lassen: `bat cd C:\tci-bridge node tci-rigctld.mjs --rigctl C:\hamlib\bin\rigctl.exe -m 3073 -r COM4 -s 115200` Die Zeile ist lang: Achten Sie darauf, dass sie wirklich mit `-s 115200` endet, sonst beendet sich `rigctl` mit *Type: rigctl --help*.

Auf beiden Systemen meldet die Brücke innerhalb einer Sekunde:

```
rig answered through rigctl
rig -> page 14280000 Hz
rig -> page mode USB
rig -> page RX
```

Öffnen Sie dann die Empfängerseite in einem Browser auf demselben Computer, öffnen Sie **TCI-CAT** und schalten Sie **CAT Sync** ein. Der Punkt wird grün und die Brücke meldet `page connected`. Drehen Sie am VFO, und der Empfänger folgt; klicken Sie in den Wasserfall, und das Funkgerät folgt (`page -> rig ... Hz`); tasten Sie das Funkgerät, und die Seite verstummt. **Strg+C** beendet die Brücke und mit ihr `rigctl`.

Alles nach `--rigctl` ist das Programm `rigctl` mit seinen eigenen Optionen — genau denen, die bei der Prüfung funktioniert haben. Das ist die Regel für jedes Funkgerät: **Wenn `rigctl ... f` die Frequenz ausgibt, funktioniert die Brücke mit denselben Optionen.**

Icom-Filter (FIL1, FIL2, FIL3)

Icom-Geräte wie der IC-7300 nehmen nicht einfach jede Filterbreite an: Sie haben drei Filter, **FIL1**, **FIL2** und **FIL3**, deren Breite jeweils im Menü des Funkgeräts eingestellt wird. Mit einer Breite würde Hamlib einen davon wählen und zugleich seine Breite überschreiben — und beim IC-7300 kann diese Breite auf dem zuvor gewählten Filter landen, was die Einstellungen des Funkgeräts durcheinanderbringt. Bei Icom-Geräten sendet die Brücke daher nie eine Breite: Sie nimmt den Filter, dessen Referenzbreite dem Durchlassbereich der Seite am nächsten liegt, und **wählt** ihn nur aus, mit dem eigenen CI-V-Befehl des Funkgeräts. Die am Funkgerät eingestellten Breiten werden nie verändert.

Betriebsart	FIL1	FIL2	FIL3
USB, LSB (und ihre Datenbetriebsarten)	2700 Hz	2400 Hz	1800 Hz
CW	1200 Hz	500 Hz	250 Hz
AM	9000 Hz	6000 Hz	3000 Hz
FM	15000 Hz	10000 Hz	7000 Hz

Beim IC-7300 (`-m 3073`) geschieht das automatisch. Stellen Sie die SSB-Filter des Funkgeräts passend ein — FIL1 2,7 kHz, FIL2 2,4 kHz, FIL3 1,8 kHz (**FIL** am Funkgerät gedrückt halten) —, dann wählt 2,7 / 2,4 / 1,8 kHz unter **IF Filters** FIL1 / FIL2 / FIL3, und die Taste **FIL** am Funkgerät setzt den Durchlassbereich der Seite auf die Breite dieses Filters. Die Brücke meldet zum Beispiel `page -> rig filter 2398 Hz LSB -> FIL2 sent`.

Option	Zweck
<code>--filters 3000,2400,1800</code>	Andere SSB-Referenzbreiten, in der Reihenfolge FIL1,FIL2,FIL3; schaltet die Filterauswahl auch für ein anderes Icom-Gerät ein
<code>--civ A4</code>	Die CI-V-Adresse des Funkgeräts in Hex, wenn sie nicht 94 ist (IC-705 A4, IC-9700 A2, IC-7610 98)
<code>--filters off</code>	Breiten stattdessen über Hamlib senden, wie bei anderen Herstellern

Beide Optionen stehen vor `--rigctl`, z. B. `node tci-rigctld.mjs --filters 3000,2400,1800 --civ A4 --rigctl rigctl -m 3085 -r /dev/ttyACM0 -s 19200`. Nur mit einem IC-7300 ausprobiert. Andere Hersteller

— Yaesu, Kenwood, Elecraft und die übrigen — erhalten die Breite der Seite über Hamlib, das sie so genau einstellt, wie das Funkgerät es erlaubt.

Andere Funkgeräte

Dieselben Schritte gelten für jedes Funkgerät, das Hamlib unterstützt; nur die Optionen ändern sich.

1. **Das Funkgerät vorbereiten.** Im Menü die CAT- (oder CI-V-)Geschwindigkeit notieren und eine Einstellung wie *CAT über USB* oder *CI-V Transceive* einschalten, falls vorhanden. Unter Windows den USB-Treiber des Herstellers installieren.
2. **Die Hamlib-Modellnummer finden** in der Liste, die `rigctl -l` ausgibt:
3. Linux: `rigctl -l | grep -i 991`
4. Windows: `C:\hamlib\bin\rigctl.exe -l | findstr /i 991`
5. **Den Port finden.** Linux: `ls /dev/serial/by-id/`. Windows: Geräte-Manager. Manche Funkgeräte legen **zwei** Ports an — Yaesu FT-991A, FTDX10 und FT-710 nennen sie *Enhanced* und *Standard*; CAT liegt auf dem **Enhanced**-Port.
6. **Prüfen, dann die Brücke starten** mit `-m <Modell> -r <Port> -s <Geschwindigkeit>`, wie im IC-7300-Beispiel: zuerst `rigctl -m ... -r ... -s ... f`, dann dieselben Optionen nach `--rigctl`.

Einige Modellnummern aus Hamlib 4.5 — prüfen Sie sie mit `rigctl -l`, da eine andere Hamlib-Version ein Gerät anders nummerieren kann:

Funkgerät	-m	Funkgerät	-m
Icom IC-7300	3073	Yaesu FT-991 / FT-991A	1035
Icom IC-705	3085	Yaesu FTDX10	1042
Icom IC-7610	3078	Yaesu FT-710	1049
Icom IC-9700	3081	Yaesu FT-891	1036
Xiegu G90	3088	Yaesu FT-817	1020
Xiegu X6100	3087	Kenwood TS-590SG	2037
Elecraft K3 / K3S	2029	Kenwood TS-890S	2041
Elecraft KX3	2045	Kenwood TS-2000	2014
Elecraft K4	2047	QRP Labs QCX / QDX	2052

Zusätzliche Optionen, nur wenn nötig:

- **Ein Icom-Gerät mit geänderter CI-V-Adresse:** `-c` mit der Adresse *dezimal* hinzufügen — 94h ist `-c 148`.
- **Eine Einstellung, die Hamlib für dieses Gerät anbietet:** `rigctl -m <Modell> -L` listet sie auf; eine setzen mit `-C name=wert`, z. B. `-C post_write_delay=10` für eine langsame Schnittstelle.
- **Ein Funkgerät, das beim Öffnen des Ports sendet oder neu startet:** Manche CAT-Kabel nutzen DTR oder RTS für PTT; `-C dtr_state=OFF -C rts_state=OFF` hinzufügen.

Beispiel: Yaesu FT-991A

Nicht mit einem echten FT-991A ausprobieren — es folgt Hamlibs Einstellungen für dieses Gerät; die Prüfung `rigctl ... f` zeigt sofort, ob es funktioniert.

Am Funkgerät **CAT RATE** (Menü 031) auf `38400` stellen. Hamlibs Modellnummer für FT-991 und FT-991A ist 1035.

Das USB-Kabel des FT-991A legt **zwei** serielle Ports an. CAT liegt auf dem **Enhanced**-Port:

- **Linux:** `ls /dev/serial/by-id/` zeigt zwei Zeilen für das Funkgerät; die auf `-if00-port0` endende ist Enhanced, meist `/dev/ttyUSB0`.
- **Windows:** Der Geräte-Manager zeigt *Silicon Labs Dual CP2105 USB to UART Bridge: Enhanced COM Port (COM5)* und einen *Standard COM Port*; verwenden Sie die Nummer des Enhanced-Ports.

Linux:

```
rigctl -m 1035 -r /dev/ttyUSB0 -s 38400 f
cd ~/PhantomSDR-Plus/tci-bridge
node tci-rigctld.mjs --rigctl rigctl -m 1035 -r /dev/ttyUSB0 -s 38400
```

Windows:

```
C:\hamlib\bin\rigctl.exe -m 1035 -r COM5 -s 38400 f
cd C:\tci-bridge
node tci-rigctld.mjs --rigctl C:\hamlib\bin\rigctl.exe -m 1035 -r COM5 -s 38400
```

Gibt die Prüfung nichts Brauchbares aus, ist meist der Standard- statt des Enhanced-Ports gewählt, oder CAT RATE weicht von `-s` ab.

Beispiel: Kenwood TS-590SG

Nicht mit einem echten TS-590SG ausprobieren — es folgt Hamlibs Einstellungen für dieses Gerät; die Prüfung `rigctl ... f` zeigt sofort, ob es funktioniert.

Im Menü des Funkgeräts die Baudrate des **USB**-Ports auf `115200` stellen (die Menünummer steht im Handbuch des TS-590SG). Hamlibs Modellnummer ist `2037`; der ältere TS-590S ist `2031`. Unter Windows zuerst Kenwoods Treiber für den virtuellen COM-Port des USB-Anschlusses installieren.

Das USB-Kabel legt **einen** seriellen Port an:

- **Linux:** die Zeile des Funkgeräts in `ls /dev/serial/by-id/`, meist `/dev/ttyUSB0`.
- **Windows:** Geräte-Manager → Anschlüsse (COM & LPT), zum Beispiel `COM6`.

Linux:

```
rigctl -m 2037 -r /dev/ttyUSB0 -s 115200 f
cd ~/PhantomSDR-Plus/tci-bridge
node tci-rigctld.mjs --rigctl rigctl -m 2037 -r /dev/ttyUSB0 -s 115200
```

Windows:

```
C:\hamlib\bin\rigctl.exe -m 2037 -r COM6 -s 115200 f
cd C:\tci-bridge
node tci-rigctld.mjs --rigctl C:\hamlib\bin\rigctl.exe -m 2037 -r COM6 -s 115200
```

Schlägt die Prüfung fehl, weicht meist die USB-Baudrate von `-s` ab, oder das Kabel steckt in der RS-232-Buchse (COM), während `-r` den USB-Port nennt.

Weitere Beispiele

Keines davon wurde mit echter Hardware ausprobiert. Jede Zeile nennt die Optionen, die bei der Prüfung nach `rigctl` und bei der Brücke nach `--rigctl rigctl` stehen; stellen Sie im Menü des Funkgeräts dieselbe Geschwindigkeit ein. Der Linux-Port ist der übliche — prüfen Sie ihn mit `ls /dev/serial/by-id/`. Unter Windows den Port durch die COM-Nummer aus dem Geräte-Manager und `rigctl` durch `C:\hamlib\bin\rigctl.exe` ersetzen.

Funkgerät	Menü des Funkgeräts	Optionen (Linux)	Hinweise
Icom IC-705 (USB)	CI-V USB Baud Rate 19200	-m 3085 -r /dev/ ttyACM0 -s 19200	Zwei Ports erscheinen; CI-V ist der erste (ttyACM0)
Icom IC-7100 (USB)	CI-V USB Baud Rate 19200	-m 3070 -r /dev/ ttyUSB0 -s 19200	Hamlibs höchste Geschwindigkeit für dieses Gerät ist 19200
Icom IC-7610	CI-V USB Baud Rate 115200	-m 3078 -r /dev/ ttyUSB0 -s 115200	
Icom IC-9700	CI-V USB Baud Rate 38400	-m 3081 -r /dev/ ttyUSB0 -s 38400	
Yaesu FTDX101D / FTDX101MP	CAT RATE 38400	-m 1040 -r /dev/ ttyUSB0 -s 38400	-m 1044 für den FTDX101MP. Zwei Ports; Enhanced verwenden
Yaesu FTDX10	CAT RATE 38400	-m 1042 -r /dev/ ttyUSB0 -s 38400	Zwei Ports; Enhanced verwenden, wie beim FT-991A
Yaesu FT-710	CAT RATE 38400	-m 1049 -r /dev/ ttyUSB0 -s 38400	Zwei Ports; Enhanced verwenden, wie beim FT-991A
Yaesu FT-891	CAT RATE 38400	-m 1036 -r /dev/ ttyUSB0 -s 38400	Zwei Ports; Enhanced verwenden, wie beim FT-991A
Yaesu FT-450D	CAT RATE 38400	-m 1046 -r /dev/ ttyUSB0 -s 38400	RS-232-Buchse: einen USB-Seriell-Adapter verwenden
Yaesu FT-817 / FT-818	CAT RATE 38400	-m 1020 -r /dev/ ttyUSB0 -s 38400	-m 1041 für den FT-818. Braucht ein CAT-Kabel an der ACC-Buchse
Yaesu FT-857 / FT-897	CAT RATE 38400	-m 1022 -r /dev/ ttyUSB0 -s 38400	-m 1023 für den FT-897. Braucht ein CAT-Kabel
Kenwood TS-890S (USB)	USB-Baudrate 115200	-m 2041 -r /dev/ ttyUSB0 -s 115200	
Kenwood TS-990S (USB)	USB-Baudrate 115200	-m 2039 -r /dev/ ttyUSB0 -s 115200	
Kenwood TS-590SG / TS-590S (USB)	USB-Baudrate 115200	-m 2037 -r /dev/ ttyUSB0 -s 115200	Ausführliches Beispiel oben
Kenwood TS-480	COM-Port-Baudrate 57600	-m 2028 -r /dev/ ttyUSB0 -s 57600	RS-232-Buchse: einen USB-Seriell-Adapter verwenden
Kenwood TS-2000	COM-Port-Baudrate 57600	-m 2014 -r /dev/ ttyUSB0 -s 57600	RS-232-Buchse; Hamlibs höchste Geschwindigkeit für dieses Gerät ist 57600
Elecraft K4 (USB)	RS232-Geschwindigkeit 115200	-m 2047 -r /dev/ ttyUSB0 -s 115200	
Elecraft K3 / K3S	RS232-Geschwindigkeit 38400	-m 2029 -r /dev/ ttyUSB0 -s 38400	K3S: USB; K3: serieller Port oder KUSB-Kabel
Elecraft KX3	RS232-Geschwindigkeit 38400	-m 2045 -r /dev/ ttyUSB0 -s 38400	KXUSB-Kabel
Elecraft KX2		-m 2044 -r /dev/ ttyUSB0 -s 38400	KXUSB-Kabel

Funkgerät	Menü des Funkgeräts	Optionen (Linux)	Hinweise
	RS232-Geschwindigkeit 38400		
Xiegu G90	CI-V-Baudrate 19200	-m 3088 -r /dev/ttyUSB0 -s 19200	Hamlib nutzt die Standard-CI-V-Adresse des G90
Xiegu X6100	CI-V-Baudrate 19200	-m 3087 -r /dev/ttyUSB0 -s 19200	Hamlibs höchste Geschwindigkeit für dieses Gerät ist 19200
Lab599 TX-500	—	-m 2050 -r /dev/ttyUSB0 -s 9600	Hamlib nutzt nur 9600
ELAD FDM-DUO	—	-m 33001 -r /dev/ttyUSB0 -s 115200	
QRP Labs QDX	—	-m 2052 -r /dev/ttyACM0 -s 9600	Ein USB-Seriell-Port; die Geschwindigkeit spielt keine Rolle, muss aber angegeben werden

Vollständiger Linux-Befehl für den IC-705, als Beispiel, wie eine Zeile zu lesen ist:

```
rigctl -m 3085 -r /dev/ttyACM0 -s 19200 f
node tci-rigctld.mjs --rigctl rigctl -m 3085 -r /dev/ttyACM0 -s 19200
```

Funkgeräte, die andere Software steuert

Manche Funkgeräte werden bereits von einem Programm gesteuert, das die Aufgabe übernehmen kann — manchmal ganz ohne Brücke. Keiner dieser Wege wurde ausprobiert.

Funkgerät und Programm	Was tun
SunSDR mit ExpertSDR2/3, FlexRadio mit AetherSDR, Apache Labs ANAN / Hermes mit Thetis	Keine Brücke. Den TCI-Server des Programms einschalten und TCI-CAT direkt verwenden — die Ports 50001 und 40001 werden automatisch gefunden
FlexRadio mit SmartSDR (Windows)	In SmartSDR CAT einen Port anlegen. Er spricht Kenwood-CAT, daher funktioniert ein serieller Port dort als TS-2000: -m 2014 -r COM8 -s 57600. Ein TCP-Port funktioniert mit Hamlibs FlexRadio-Modell: -m 2036 -r 127.0.0.1:<Port>
Jedes von flrig gesteuerte Funkgerät	flrig laufen lassen und Hamlibs flrig-Modell verwenden; das Funkgerät bleibt mit fldigi, WSJT-X und Logbüchern geteilt: -m 4 -r 127.0.0.1:12345
Ein laufender rigctld oder ein Programm mit einem <i>Hamlib NET rigctl</i> -Server	Die Brücke ohne --rigctl mit dieser Adresse starten: node tci-rigctld.mjs 50001 127.0.0.1:4532

Wie überall stehen die Optionen bei der Prüfung nach rigctl und bei der Brücke nach --rigctl rigctl. Eine Netzwerkadresse wie 127.0.0.1:12345 braucht kein -s.

Das Funkgerät teilen: der rigctld-Weg

Wenn WSJT-X oder ein Logbuch das Funkgerät benutzen soll, während die Brücke läuft, starten Sie Hamlibs Server rigctld mit denselben Optionen, lassen ihn laufen und verbinden beide Programme damit:

```
rigctld -m 3073 -r /dev/ttyUSB0 -s 115200
node tci-rigctld.mjs
```

Ohne `--rigctl` sucht die Brücke `rigctld` unter `127.0.0.1:4532` und meldet `rigctld connected`; `node tci-rigctld.mjs 50001 192.168.1.50:4532` nutzt einen `rigctld` auf einem anderen Computer. In WSJT-X das Rig *Hamlib NET rigctl* unter derselben Adresse wählen. Unter Windows ist der Server `C:\hamlib\bin\rigctld.exe` mit denselben Optionen; antwortet Windows mit *Access is denied*, nehmen Sie `--rigctl` und schließen das andere Programm, solange die Brücke läuft.

Ports. Die Brücke stellt TCI auf Port 50001 bereit; ein anderer Port steht als Erstes in der Befehlszeile, z. B. `node tci-rigctld.mjs 40001 --rigctl ...`. Laufen Brücke und Browser auf verschiedenen Computern, tragen Sie die Adresse des Brücken-Computers unter **Host** ein.

Fehlerbehebung bei TCI-CAT

Symptom	Wahrscheinliche Ursache	Was tun
Punkt bleibt grau	TCI-Server oder Brücke läuft nicht; falscher Host ; die Erlaubnis des Browsers für das lokale Netz wurde verweigert	TCI in ExpertSDR/Thetis/AetherSDR aktivieren oder die Brücke starten; Host prüfen; lokalen Netzzugriff in den Website-Einstellungen erlauben
<code>rigctl ... f</code> meldet einen Fehler oder wartet	Falscher Port oder falsche Geschwindigkeit; keine <code>dialout</code> -Berechtigung (Linux) oder kein USB-Treiber (Windows); ein anderes Programm belegt den Port	<code>-s</code> an das Menü des Funkgeräts anpassen; Port prüfen (<code>ls /dev/serial/by-id/</code> , Geräte-Manager); andere Funkgeräteprogramme schließen
Brücke meldet <code>rigctl stopped ... Type: rigctl --help</code>	Der Befehl ist unvollständig — meist fehlt die Geschwindigkeit nach <code>-s</code>	Die ganze Zeile neu eingeben
Brücke meldet <code>The value after -s is missing</code>	Die Befehlszeile wurde beim Einfügen abgeschnitten	Das Ende der Zeile neu eingeben, z. B. <code>-s 115200</code>
Der Filter folgt in einer Richtung nicht	CAT Sync ist aus, auf der Seite läuft ein Decoder, oder (IC-7300) die FIL-Breiten des Funkgeräts weichen von 2700 / 2400 / 1800 Hz ab	CAT Sync einschalten; den Decoder beenden; die FIL-Breiten am Funkgerät einstellen oder <code>--filters</code> mit den Breiten des Funkgeräts angeben
Brücke meldet <code>rigctl stopped ... rig_open: error</code>	<code>rigctl</code> kann den Port nicht öffnen	Wie bei <code>rigctl ... f</code> oben
Brücke meldet <code>the rig is not answering</code>	Der Port öffnet sich, aber das Funkgerät antwortet nicht: falsche Geschwindigkeit oder falsches Modell, CAT im Menü ausgeschaltet oder geänderte Icom-CI-V-Adresse	Die Prüfung <code>rigctl ... f</code> wiederholen; bei geänderter CI-V-Adresse <code>-c</code> hinzufügen
Brücke meldet <code>rigctld not reachable</code>	Ohne <code>--rigctl</code> gestartet, und kein <code>rigctld</code> läuft	<code>--rigctl ...</code> hinzufügen oder zuerst <code>rigctld</code> starten
<code>rigctld.exe</code> meldet <code>Access is denied</code> (Windows)	Windows verweigert die Ausführung	<code>--rigctl</code> verwenden — dafür genügt <code>rigctl.exe</code>
Verbunden, aber der Empfänger bewegt sich nicht	CAT Sync ist aus	Einschalten — die Stummschaltung beim Senden wirkt auch ohne
Betriebsart des Funkgeräts folgt nicht	Eine Betriebsart ohne Entsprechung im Empfänger, oder der Hamlib-Treiber meldet einen ungewöhnlichen Namen	Die Frequenz wird trotzdem synchronisiert; Betriebsart auf der Seite einstellen
Keine Stummschaltung beim Senden	Das Funkgerät oder sein Hamlib-Treiber meldet den Sendezustand nicht	Auf der Seite ist nichts einzustellen
Die Skala springt hin und her	Desktop PhantomSDR+ oder ein anderes Programm synchronisiert das Funkgerät ebenfalls	Immer nur eine Steuerung verwenden

Für Empfängerbetreiber

Nichts einzustellen. Die Transceiver-Steuerung nutzt eine kleine JavaScript-Schnittstelle, die jede PhantomSDR-Plus-Seite bereits enthält; sie braucht keine Servereinstellung, keinen offenen Port und keine Admin-Berechtigung. Die Filter- und Stummschaltfunktionen kamen mit Version 4.1.0 — danach das Frontend neu bauen (`./recompile.sh`, Option 2); der Empfänger muss nicht angehalten werden. Auch KiwiSDR-, WebSDR- und UberSDR-Empfänger brauchen nichts: Die Anwendung nutzt die Bedienelemente, die ihre Seiten schon haben.

Die Schaltfläche **TCI-CAT** kam ebenfalls mit 4.1.0, wieder nur ein Neubau des Frontends. Sie öffnet keinen Port auf dem Empfänger: Die Verbindung läuft vom Browser jedes Hörers zu seinem eigenen Computer. Die Hamlib-Brücke `tci-bridge/tci-rigctld.mjs` ist für Hörer zu Hause gedacht; der Empfänger benutzt sie nicht.

Für Entwickler: die Seitenschnittstelle

Desktop PhantomSDR+ und das CATsync Tool nutzen beide diese Funktionen, die jede PhantomSDR-Plus-Seite nach dem Laden auf `window` bereitstellt (KiwiSDR-, WebSDR- und UberSDR-Seiten werden über ihre eigenen, anderen Bedienelemente gesteuert):

Funktion	Rückgabe / Wirkung
<code>catsync_ready</code>	<code>true</code> , sobald die Funktionen unten installiert sind
<code>catsync_getFrequency()</code>	Abgestimmte Frequenz, Hz
<code>catsync_setFrequency(hz)</code>	Auf <code>hz</code> abstimmen
<code>catsync_getMode()</code>	USB, LSB, CW, CW-L, AM, QUAM, FM, WBFM, RADEU, RADEL
<code>catsync_setMode(mode)</code>	Betriebsart setzen; setzt den Durchlassbereich auf den Standard der Betriebsart zurück
<code>catsync_getBandwidth()</code>	Gesamte Durchlassbreite, Hz
<code>catsync_setBandwidth(hz)</code>	Breite setzen — wächst in USB nach oben, in LSB nach unten, sonst gleichmäßig. Nach <code>catsync_setMode</code> aufrufen
<code>catsync_getMute()</code>	<code>true</code> , wenn stummgeschaltet
<code>catsync_setMute(on)</code>	Stumm schalten oder aufheben, über die Stummschalttaste der Seite

Die letzten vier kamen mit 4.1.0, daher vor dem Aufruf prüfen:

```
if (window.catsync_ready) {
  window.catsync_setFrequency(7074000)
  window.catsync_setMode('USB')
  if (typeof window.catsync_setBandwidth === 'function') window.catsync_setBandwidth(2400)
}
```

Das Setzen der Frequenz stimmt den Ton neu ab; rufen Sie einen Setter also nur auf, wenn sich der Wert tatsächlich geändert hat — ein Setter, der ständig mit demselben Wert aufgerufen wird, ist hörbar. Die älteren Einstiegspunkte im KiwiSDR/WebSDR-Stil (`setfreq`, `set_mode`, `freqset_complete`) sind für Werkzeuge, die sie erwarten, weiterhin vorhanden.

Fehlerbehebung

Symptom	Wahrscheinliche Ursache	Was tun
<i>Could not open ttyUSB0</i> (Linux)	Nicht in der Gruppe <code>dialout</code> , oder ein anderes Programm hat die Schnittstelle	<code>sudo usermod -aG dialout \$USER</code> , ab- und anmelden; WSJT-X, Logbücher, Geräte-Tools schließen
Die Empfängeranzeige sagt <i>not a receiver this app can drive</i>	Eine andere Art von Web-Empfänger, oder die Seite lädt noch	Unterstützt werden PhantomSDR-Plus, KiwiSDR, PA3FWM WebSDR und UberSDR; einer langsamen Seite ein paar Sekunden geben
<i>The rig did not answer</i>	Falsche Geschwindigkeit, falscher Gerätetyp, falsche CI-V-Adresse, Gerät aus	Geschwindigkeit an das Gerätemenü anpassen; CI-V-Adresse prüfen; Hamlib versuchen
<i>Hamlib is not installed</i>	Kein <code>rigctld</code> gefunden	Linux: <code>sudo apt install libhamlib-utils</code> . Oder rigctld program auf seinen Pfad setzen
<i>rigctld stopped: ...</i>	Hamlib konnte das Gerät nicht öffnen — seine eigene Meldung folgt	Meist Schnittstelle oder Geschwindigkeit; der Text nach dem Doppelpunkt ist Hamlibs Begründung
<i>flrig is not running at ...</i>	flrig geschlossen, oder sein XML-RPC-Port weicht ab	flrig starten; den Port in der flrig-Konfiguration prüfen
Verbunden, aber der Empfänger bewegt sich nicht	Kein Stationsfenster offen, oder Receiver window auf eine geschlossene Station festgelegt	Station öffnen oder <i>The station window last in front</i> wählen
Das Gerät sendet beim Verbinden	DTR oder RTS tastet das Gerät über Ihr Interface	Häkchen bei DTR on und RTS on entfernen
Filter folgt nicht	Empfänger älter als 4.1.0, oder integrierter Kenwood-/Yaesu-Treiber	Frequenz und Betriebsart synchronisieren weiter; für den Filter bei Kenwood/Yaesu Hamlib nutzen
Stummschaltung beim Senden wirkt nicht	Empfänger ohne Update, oder das Gerät meldet den Sendezustand nicht	Wie oben
<i>Lost the rig ... reconnecting</i>	Kabel gezogen, Gerät ausgeschaltet oder <code>rigctld</code> beendet	Nichts — alle 3 Sekunden neuer Versuch, danach geht es weiter
Beide Seiten springen ständig	Zwei Programme steuern das Gerät gleichzeitig	Nur ein Programm das Gerät einstellen lassen oder über flrig teilen

Bekannte Einschränkungen

- Ein Funkgerät, ein Empfängerfenster zur selben Zeit.
- Andere Empfänger als PhantomSDR-Plus, KiwiSDR, PA3FWM WebSDR und UberSDR — etwa OpenWebRX — werden nicht unterstützt.
- Die integrierten Treiber folgen den veröffentlichten Protokollen der Hersteller und wurden gegen simulierte Geräte und echtes Hamlib getestet; für ein Gerät, das sich anders verhält, ist Hamlib die Ausweichlösung.
- Split-Betrieb, VFO B, RIT/XIT und Speicherkanäle werden nicht synchronisiert — nur die Frequenz des aktiven VFO.

- TCI-CAT auf der Empfängerseite synchronisiert weder Split noch VFO B, RIT/XIT oder Transverter-Versatz und wurde nur mit einem IC-7300 über die Hamlib-Brücke ausprobiert.
- Unter Linux verwenden die Pakete das Hamlib der Distribution; keines ist mitgeliefert.

KAPITEL 16

Projektstruktur

PROJECT_STRUCTURE.md

Projektstruktur von PhantomSDR-Plus

Dieses Dokument gibt einen umfassenden Überblick über die Verzeichnisstruktur von PhantomSDR-Plus, die Dateiorganisation und die Beziehungen zwischen den Komponenten.

Inhaltsverzeichnis

1. Verzeichnisbaum
 2. Wurzelverzeichnis
 3. Quellcode (`src/`)
 4. Frontend (`frontend/`)
 5. Frequenzlisten (`frequencylist/`)
 6. Konfigurationsdateien
 7. Build-System
-

Verzeichnisbaum

PhantomSDR-Plus

```

├─ add-receiver.sh          # fügt einen weiteren Empfänger hinzu: Treiber, instances/<name>/,
receivers.toml – siehe docs/MULTI_RECEIVER.md
├─ admin_config.json
├─ admin_server.py
├─ autorun
│  ├─ audiotap.js
│  ├─ bandplan.js
│  ├─ decodeworker.js
│  ├─ index.js
│  ├─ manager.js
│  ├─ package.json
│  ├─ pool.js
│  ├─ probe-ft8.js
│  ├─ probe-js8.js
│  ├─ pskreporter.js
│  ├─ spotparse.js
│  ├─ wasm-shim.js
│  └─ wsprnet.js
├─ cf32_to_real.c          # Fobos HF: behält I aus dem CF32 von rx_sdr, auf s16 skaliert
(gebaut von setup-fobos.sh)
├─ chat_history.txt
├─ check-go.sh             # alt: Watchdog der go.sh-Kette
├─ config-airspyhf.toml
├─ config.example.hackrf.toml
├─ config.example.rtlsdr.toml
├─ config-fobos-hf.toml
├─ config-fobos.toml
├─ config-hackrf.toml
├─ config-rspla.toml
├─ config-rtl.toml
├─ config-rx888mk2.toml
├─ config.toml
├─ connection_impl.hpp
├─ docs
│  ├─ ADMIN_PANEL_SETUP.md
│  ├─ Aether_config.md
│  ├─ CONNECTION_LIMITS.md
│  └─ de
│     ├─ ADMIN_PANEL_SETUP.md
│     ├─ Aether_config.md
│     ├─ CONNECTION_LIMITS.md
│     ├─ DECODERS.md
│     ├─ EDITING_VARIANTS.md
│     ├─ INSTALLATION.md
│     ├─ MULTI_RECEIVER.md
│     ├─ PhantomSDR-Plus-Dokumentation-DE.pdf
│     ├─ PROJECT_STRUCTURE.md
│     ├─ RADE_General_INSTALL_MANUAL_LINUX.md
│     ├─ RADE_README.md
│     ├─ README.md
│     ├─ RECEIVE_DIVERSITY.md
│     ├─ RIG_CONTROL.md
│     ├─ THERMAL_GUARD.md
│     └─ USER_GUIDE.md
├─ DECODERS.md
├─ EDITING_VARIANTS.md
└─ el
   ├─ ADMIN_PANEL_SETUP.md
   ├─ Aether_config.md
   ├─ CONNECTION_LIMITS.md
   ├─ DECODERS.md
   ├─ EDITING_VARIANTS.md
   └─ INSTALLATION.md

```

- └─ MULTI_RECEIVER.md
- └─ PhantomSDR-Plus-Documentation-EL.pdf
- └─ PROJECT_STRUCTURE.md
- └─ RADE_General_INSTALL_MANUAL_LINUX.md
- └─ RADE_README.md
- └─ README.md
- └─ RECEIVE_DIVERSITY.md
- └─ RIG_CONTROL.md
- └─ THERMAL_GUARD.md
- └─ USER_GUIDE.md
- └─ es
 - └─ ADMIN_PANEL_SETUP.md
 - └─ Aether_config.md
 - └─ CONNECTION_LIMITS.md
 - └─ DECODERS.md
 - └─ EDITING_VARIANTS.md
 - └─ INSTALLATION.md
 - └─ MULTI_RECEIVER.md
 - └─ PhantomSDR-Plus-Documentation-ES.pdf
 - └─ PROJECT_STRUCTURE.md
 - └─ RADE_General_INSTALL_MANUAL_LINUX.md
 - └─ RADE_README.md
 - └─ README.md
 - └─ RECEIVE_DIVERSITY.md
 - └─ RIG_CONTROL.md
 - └─ THERMAL_GUARD.md
 - └─ USER_GUIDE.md
- └─ fr
 - └─ ADMIN_PANEL_SETUP.md
 - └─ Aether_config.md
 - └─ CONNECTION_LIMITS.md
 - └─ DECODERS.md
 - └─ EDITING_VARIANTS.md
 - └─ INSTALLATION.md
 - └─ MULTI_RECEIVER.md
 - └─ PhantomSDR-Plus-Documentation-FR.pdf
 - └─ PROJECT_STRUCTURE.md
 - └─ RADE_General_INSTALL_MANUAL_LINUX.md
 - └─ RADE_README.md
 - └─ README.md
 - └─ RECEIVE_DIVERSITY.md
 - └─ RIG_CONTROL.md
 - └─ THERMAL_GUARD.md
 - └─ USER_GUIDE.md
- └─ hr
 - └─ ADMIN_PANEL_SETUP.md
 - └─ Aether_config.md
 - └─ CONNECTION_LIMITS.md
 - └─ DECODERS.md
 - └─ EDITING_VARIANTS.md
 - └─ INSTALLATION.md
 - └─ MULTI_RECEIVER.md
 - └─ PhantomSDR-Plus-Documentation-HR.pdf
 - └─ PROJECT_STRUCTURE.md
 - └─ RADE_General_INSTALL_MANUAL_LINUX.md
 - └─ RADE_README.md
 - └─ README.md
 - └─ RECEIVE_DIVERSITY.md
 - └─ RIG_CONTROL.md
 - └─ THERMAL_GUARD.md
 - └─ USER_GUIDE.md
- └─ INSTALLATION.md
- └─ MULTI_RECEIVER.md
- └─ PhantomSDR-Plus-Documentation-EN.pdf
- └─ PROJECT_STRUCTURE.md

```

├── RADE_General_INSTALL_MANUAL_LINUX.md
├── RADE_README.md
├── README.md
├── RECEIVE_DIVERSITY.md
├── RIG_CONTROL.md
├── ru
│   ├── ADMIN_PANEL_SETUP.md
│   ├── Aether_config.md
│   ├── CONNECTION_LIMITS.md
│   ├── DECODERS.md
│   ├── EDITING_VARIANTS.md
│   ├── INSTALLATION.md
│   ├── MULTI_RECEIVER.md
│   ├── PhantomSDR-Plus-Documentation-RU.pdf
│   ├── PROJECT_STRUCTURE.md
│   ├── RADE_General_INSTALL_MANUAL_LINUX.md
│   ├── RADE_README.md
│   ├── README.md
│   ├── RECEIVE_DIVERSITY.md
│   ├── RIG_CONTROL.md
│   ├── THERMAL_GUARD.md
│   └── USER_GUIDE.md
├── sdr-stats
│   ├── package.json
│   ├── readme_de.md
│   ├── readme_el.md
│   ├── readme_es.md
│   ├── readme_fr.md
│   ├── readme_hr.md
│   ├── README.md
│   └── readme_ru.md
├── THERMAL_GUARD.md
├── USER_GUIDE.md
├── websdr2.png
├── websdr3.png
├── websdr.png
├── favicon.ico
├── fftw_wisdom
├── fix_local_geo.py
├── frequencylist
│   ├── 0.TXT
│   ├── A26all00.TXT          # nicht im Repo: von update-markers.sh aus a26allx2.zip entpackt
│   ├── a26allx2.zip
│   ├── admin.txt
│   ├── antenna.txt
│   ├── broadcas.txt
│   ├── curl-output.txt
│   ├── currentUpdateFile.txt
│   ├── fmorg.txt
│   ├── generate-current-shortwave.py
│   ├── language.txt
│   ├── mymarkers.json
│   ├── README.md
│   ├── shortwavestations.json
│   ├── site.txt
│   └── update-markers.sh
├── frontend
│   ├── build-all.sh
│   ├── build-default.sh
│   ├── build-mobile.sh
│   ├── debug-title.sh
│   ├── favicon.ico
│   ├── fix-title-python.py
│   ├── index.html
│   └── jsconfig.json

```

```

├─ LICENSE
├─ make-redirect-stubs.sh
├─ mobile
│   └─ index.html
├─ package.json
├─ package-lock.json
├─ pnpm-lock.yaml
├─ postcss.config.cjs
├─ .prettierrc.json      # Prettier-Formatierungsregeln für die Frontend-Quellen
├─ public
│   ├── analyze_users.py
│   ├── decoders
│   │   ├── ft8_lib.wasm
│   │   ├── js8_dict.bin  # JS8-Wörterbuch, erst bei Bedarf geladen
│   │   └─ js8.wasm      # JS8-Decoder, gebaut aus jsdsp/js8_wasm
│   ├── logo.jpg
│   ├── stats.html
│   ├── users.html
│   └─ wf-message.json
├─ README.md
├─ site_information.json
├─ src
│   ├── app.css
│   ├── App.svelte
│   ├── assets
│   │   ├── amateurfrequencies.json
│   │   ├── background.jpg
│   │   ├── shortwavestations.json
│   │   ├── SSTV.png
│   │   ├── SSTV.svg
│   │   └─ svelte.png
│   ├── audio.js
│   ├── audio-stream-worklet.js
│   ├── bands-config.js
│   ├── broadcastSchedules.js
│   ├── clientVersion.js
│   ├── cwDecoder.js
│   ├── cw.worker.js
│   ├── cwWorkerProxy.js
│   ├── decoder.worker.js
│   ├── diversity.js
│   ├── diversityList.js
│   ├── eventBus.js
│   ├── events.js
│   ├── fax.js
│   ├── fax.worker.js
│   ├── faxWorkerProxy.js
│   ├── fft.js
│   ├── flacLowLatency.js      # FLAC-Decoder, der jedes Paket beim Eintreffen abspielt
│   ├── fsk.js
│   ├── fsk.worker.js
│   ├── fskWorkerProxy.js
│   ├── kiwiSource.js
│   ├── lib
│   │   ├── backend.js
│   │   ├── BandSelector.svelte
│   │   ├── catsync.js
│   │   ├── chatReply.js      # Markierung und Zuordnung von Chat-Antworten
│   │   ├── CheckButton.svelte
│   │   ├── colormaps.js
│   │   ├── Counter.svelte
│   │   ├── DiversityPanel.svelte
│   │   ├── fftRadix2.js
│   │   ├── freedv-reporter.js
│   │   └─ FreeDVReporter.svelte

```

```

├─ FrequencyInput.svelte
├─ FrequencyMarkers.svelte
├─ FtxSpectrum.svelte
├─ hammeractions.js
├─ LineThroughButton.svelte
├─ Logger.svelte
├─ MagicEyeIndicator.svelte
├─ ModeIdChip.svelte
├─ ModesSelector.svelte
├─ opusMlDecoder.js
├─ PassbandTuner.svelte
├─ Popover.svelte
├─ QrssPanel.svelte
├─ receivers.js          # Daten der Empfängerauswahl (/receivers.json)
├─ rnnoise.js           # lädt RNNoise für die KI-Rauschunterdrückung
├─ rx.js                 # zu welchem Empfänger eine Seite gehört (?rx=)
├─ SMeterAnalog.svelte
├─ SMeterDigital.svelte
├─ Spectrogram.svelte
├─ StatusIndicators.svelte
├─ storage.js
├─ sUnits.js             # S-Stufen-Skala je Frequenz (S9 = -93 dBm oberhalb 30 MHz)
├─ Tooltip.svelte
├─ VersionSelector.svelte
├─ VideoAreaSelector.svelte
├─ wrappers.js
├─ main.js
├─ mobile
│   └─ backend.js
│   └─ bookmarks.js
│   └─ main.js
│   └─ Mobile.svelte
│   └─ tuning.js
├─ modeId.js
├─ modeId.worker.js
├─ modeIdWorkerProxy.js
├─ modePriors.js
├─ modules
│   └─ decode.wasm
│   └─ encode.wasm
│   └─ ft4.js
│   └─ ft8.js
│   └─ js8-decoder.js
│   └─ js8-format.js
│   └─ js8.js
│   └─ js8-reassembler.js
│   └─ js8-slots.js
│   └─ js8-tables.js
│   └─ package.json
│   └─ phantomsdrdsp_bg_fallback.js
│   └─ phantomsdrdsp_bg.js
│   └─ phantomsdrdsp_bg.wasm
│   └─ phantomsdrdsp_bg.wasm.d.ts
│   └─ phantomsdrdsp.d.ts
│   └─ phantomsdrdsp.js
│   └─ phantomsdrdsp_router.js
│   └─ wspr.js
├─ olivia.js
├─ playoutControl.js     # misst die Verbindung, hält den Wiedergabepuffer klein
├─ psk31.js
├─ refused.js
├─ remoteSource.js
├─ scanner.js
├─ siteInfo.js           # site_information.json, überlagert mit der eigenen Kopie des

```

Empfängers

```

├── sstv.js
├── sstv.worker.js
├── sstvWorkerProxy.js
├── uberSource.js
├── unused
│   ├── AudioProcessor.js
│   ├── decoder.js
│   ├── decoding.js
│   └── modules-emscripthen
│       ├── dav1d.js
│       ├── dav1dnoWasm.js
│       ├── dav1dnoWasm.js.mem
│       ├── dav1d.wasm
│       ├── decode_ft8.js
│       ├── decode_ft8.wasm
│       ├── FoxenFlac.js
│       ├── jsDSP.js
│       ├── jsDSPnoWasm.js
│       ├── jsDSPnoWasm.js.mem
│       ├── jsDSPnoWasm.wasm
│       ├── jsDSP.wasm
│       ├── libzstd.js
│       ├── LiquidDSP.js
│       ├── opus.js
│       ├── opusnoWasm.js
│       ├── opusnoWasm.js.mem
│       ├── opus.wasm
│       ├── redsea.js
│       └── redsea.wasm
├── unused.js
├── wrappers.js
├── videoRecorder.js
├── vite-env.d.ts
├── waterfall.js
├── webSdrCodec.js
├── webSdrSource.js
├── stats.html
├── svelte.config.js
├── tailwind.config.cjs
├── vite.config.js
├── go.sh # alte Startkette, ersetzt durch start-<radio>.sh
├── install_arch.sh
├── install_fedora.sh
├── install_opensuse.sh
├── install_rade.sh
├── install_rade_ubuntu22.sh
├── install.sh
├── install-stats-server.sh
├── instances # weitere Empfänger, je ein Ordner (nicht in git) – siehe docs/
├── MULTI_RECEIVER.md
├── instructions-for-airspy
├── instructions-for-rspla
├── jsdsp
│   ├── compilejs.sh
│   ├── configureredsea.sh
│   ├── extract_EXPORTED_FUNCTIONS.js
│   ├── ft8_wasm
│   │   ├── build_ft8_wasm.sh
│   │   ├── README.md
│   │   └── wasm_wrapper.c
│   └── include
│       ├── avif
│       │   ├── avif.h
│       │   └── internal.h
│       └── liquid

```

```

├── liquid.h
├── js8_wasm          # JS8-Decoder in C, als WebAssembly gebaut, mit Tests und Referenz-
Prüfstand - siehe dessen README
├── build_js8_wasm.sh
├── check_against_reference.sh
├── extract_js8_constants.py
├── extract_js8_varicode.py
├── js8_ab_test.mjs
├── js8_autorun_test.mjs
├── js8_chain_test.mjs
├── js8_constants.c
├── js8_constants.h
├── js8_decode.c
├── js8_decode.h
├── js8_decode_test.c
├── js8_encode.c
├── js8_encode.h
├── js8_frames_test.mjs
├── js8_osd.c
├── js8_osd.h
├── js8_reassembly_test.mjs
├── js8_roundtrip.c
├── js8_slots_test.mjs
├── js8_subtract.c
├── js8_subtract.h
├── js8_wasm_test.mjs
├── js8_wasm_wrapper.c
├── README.md
├── reference
├──   ├── build_in_container.sh
├──   ├── build_reference_decoder.sh
├──   ├── build_reference_frames.sh
├──   ├── Containerfile
├──   ├── Containerfile.decoder
├──   ├── ref_decode.f90
├──   ├── ref_decode.txt
├──   ├── ref_frames.cpp
├──   └── ref_frames.txt
├── ref_gen.f90
├── run_tests.sh
├── lib
├──   ├── ANR.c
├──   ├── arm_funcs.h
├──   ├── CMSIS_DSP
├──   ├──   ├── BUILDING.txt
├──   └──   └── LICENSE.txt
├──   ├── dav1d.cpp
├──   ├── NB.c
├──   ├── NR_spectral.c
├──   └── types.h
├── redsea.js
├── redsea.wasm
├── src
├──   ├── index.js
├──   ├── libzstd.js
├──   ├── LiquidDSP.js
├──   ├── NoiseProcessing.js
├──   └── wbfmpll.cpp
├── kill.sh          # alt: beendet die Serverprozesse, von go.sh aufgerufen
├── kiwi_install.sh  # installiert die KiwiSDR-Brücke in einen bestehenden Baum
├── LICENSE
├── logproxy         # rotierte Kopien der Panel-/Proxy-/Autorun-Logs
├── logrotate
├──   └── phantomsdr  # logrotate-Konfiguration für proxy.log + admin.log (Pfade vor der
Installation anpassen)

```

```

├─ manage_admin.sh
├─ markers.json
├─ meson.build
├─ meson_options.txt
├─ phantom_fftw_wisdom
├─ phantomsdr-admin.service # Beispiel-systemd-Unit für das Admin-Panel (Start beim Booten,
Neustart nach Absturz)
├─ phantomsdr-proxy.service # Beispiel-systemd-Unit für den Proxy, zusammen mit der Panel-Unit
zu installieren
├─ phantomsdr-websdr-relay.service # Beispiel-systemd-Unit für das WebSDR-Diversity-Relay
├─ proxy.py
├─ rade_helper.py
├─ rade_loadtest.csv # Ausgabe von rade_loadtest.py (eine Zeile pro Laststufe)
├─ rade_loadtest.py # RADE-Lasttest: wie viele gleichzeitige Decoder die Maschine trägt
- siehe docs/RADE_README.md
├─ rade.sh
├─ README.md
├─ recompile.sh
├─ receivers.toml.example # Empfängerliste für proxy.py – nach receivers.toml kopieren
├─ _relaunch.sh # alt: verzögerter Neustart-Helfer der go.sh-Kette
├─ request.hpp
├─ setup_admin.sh
├─ setup-airspyhf.sh # Treiberkette Airspy HF+: libairspyhf + SoapyAirspyHF + rx_sdr +
udev
├─ setup-cpufreq-perms.sh # gibt der Gruppe Schreibrecht auf scaling_max_freq, damit der
Wächter ohne root drosseln kann
├─ setup-firewall.sh # optionaler nftables-Flutschutz – siehe docs/CONNECTION_LIMITS.md
├─ setup-fobos.sh # Treiberkette Fobos: libfobos + SoapyFobosSDR + rx_sdr +
cf32_to_real + udev
├─ setup-hackrf.sh # HackRF: Paket hackrf der Distribution + udev (nichts gebaut)
├─ setup-rspla.sh # Treiberkette RSP1A: libmirisdr-5 + SoapyMiri + rx_sdr + msi2500-
Sperre + udev
├─ setup-rtlsdr.sh # RTL-SDR-Treiber: Paket der Distribution oder Blog-V4-Treiber aus
dem Quelltext + DVB-Sperre + udev
├─ setup-rx888-udev.sh
├─ setup-sdr-common.sh # gemeinsame Hilfsfunktionen der drei setup-Skripte oben
├─ setup_websdr_relay.sh # installiert das WebSDR-Diversity-Relay (Port, Identität, systemd)
├─ smeter_theme.sh
├─ src
├─ │ audio.cpp
├─ │ audio.h
├─ │ chat.cpp
├─ │ chat.h
├─ │ client.cpp
├─ │ client.h
├─ │ compression.cpp
├─ │ compression.h
├─ │ crash_handler.cpp
├─ │ crash_handler.h
├─ │ events.cpp
├─ │ events.h
├─ │ fft.cpp
├─ │ fft_cuda.cu
├─ │ fft.h
├─ │ fft_impl.cpp
├─ │ fft_mkl.cpp
├─ │ http.cpp
├─ │ kiwi_bridge.h # KiwiSDR-Protokollbrücke – siehe docs/Aether_config.md
├─ │ listing
├─ │ │ software_info.cpp
├─ │ │ software_info.h
├─ │ samplerreader.cpp
├─ │ samplerreader.h
├─ │ signal.cpp
├─ │ signal.h

```

```

├── spectrumserver.cpp
├── spectrumserver.h
├── utils
│   ├── audioprocessing.cpp
│   ├── audioprocessing.h
│   ├── dsp.cpp
│   └── dsp.h
├── utils.cpp
├── utils.h
├── waterfallcompression.cpp
├── waterfallcompression.h
├── waterfall.cpp
├── waterfall.h
├── websocket.cpp
├── websocket.h
├── start-airspyhf.sh
├── start-all.sh                # startet jeden Empfänger aus receivers.toml
├── start-fobos-hf.sh
├── start-fobos.sh
├── start-hackrf.sh
├── start-rspla.sh
├── start-rtl.sh
├── start-rx888mk2.sh
├── stop-websdr.sh
├── subprojects
│   ├── fftw3.wrap
│   ├── flac.wrap
│   ├── glaze.wrap
│   ├── libcds.wrap
│   ├── libflac.wrap
│   ├── libvolk.wrap
│   ├── ogg.wrap
│   ├── opus.wrap
│   ├── tomlplusplus-3.4.0
│   │   ├── CHANGELOG.md
│   │   ├── cmake
│   │   │   ├── install-rules.cmake
│   │   │   ├── project-is-top-level.cmake
│   │   │   ├── tomlplusplusConfig.cmake
│   │   │   ├── tomlplusplusConfig.cmake.meson.in
│   │   │   ├── tomlplusplusConfigVersion.cmake.meson.in
│   │   │   └── variables.cmake
│   │   ├── CMakeLists.txt
│   │   ├── CODE_OF_CONDUCT.md
│   │   ├── CONTRIBUTING.md
│   │   ├── cpp.hint
│   │   └── docs
│   │       ├── images
│   │       │   ├── badge-awesome.svg
│   │       │   ├── badge-C++17.svg
│   │       │   ├── badge-gitter.svg
│   │       │   ├── badge-license-MIT.svg
│   │       │   ├── badge-TOML.svg
│   │       │   ├── banner.ai
│   │       │   ├── banner.png
│   │       │   ├── banner.svg
│   │       │   ├── favicon.ico
│   │       │   ├── logo.ai
│   │       │   └── logo.svg
│   │       ├── pages
│   │       │   └── main_page.md
│   │       └── poxy.toml
│   └── examples
│       ├── benchmark_data.toml
│       └── CMakeLists.txt

```

```

├─ error_printer.cpp
├─ error_printer.vcxproj
├─ examples.hpp
├─ example.toml
├─ merge_base.toml
├─ merge_overrides.toml
├─ meson.build
├─ parse_benchmark.cpp
├─ parse_benchmark.vcxproj
├─ simple_parser.cpp
├─ simple_parser.vcxproj
├─ toml_generator.cpp
├─ toml_generator.vcxproj
├─ toml_merger.cpp
├─ toml_merger.vcxproj
├─ toml_to_json_transcoder.cpp
├─ toml_to_json_transcoder.vcxproj
├─ include
├─ meson.build
├─ toml++
├─ impl
├─ array.hpp
├─ array.inl
├─ at_path.hpp
├─ at_path.inl
├─ date_time.hpp
├─ formatter.hpp
├─ formatter.inl
├─ forward_declarations.hpp
├─ header_end.hpp
├─ header_start.hpp
├─ json_formatter.hpp
├─ json_formatter.inl
├─ key.hpp
├─ make_node.hpp
├─ node.hpp
├─ node.inl
├─ node_view.hpp
├─ parse_error.hpp
├─ parse_result.hpp
├─ parser.hpp
├─ parser.inl
├─ path.hpp
├─ path.inl
├─ preprocessor.hpp
├─ print_to_stream.hpp
├─ print_to_stream.inl
├─ simd.hpp
├─ source_region.hpp
├─ std_except.hpp
├─ std_initializer_list.hpp
├─ std_map.hpp
├─ std_new.hpp
├─ std_optional.hpp
├─ std_string.hpp
├─ std_string.inl
├─ std_utility.hpp
├─ std_variant.hpp
├─ std_vector.hpp
├─ table.hpp
├─ table.inl
├─ toml_formatter.hpp
├─ toml_formatter.inl
├─ unicode_autogenerated.hpp
├─ unicode.hpp

```

```

├── unicode.inl
├── value.hpp
├── version.hpp
├── yaml_formatter.hpp
├── yaml_formatter.inl
├── toml.h
├── toml.hpp
├── LICENSE
├── meson.build
├── meson_options.txt
├── README.md
├── src
│   ├── meson.build
│   └── toml.cpp
├── tests
│   ├── at_path.cpp
│   ├── conformance_burntsushi_invalid.cpp
│   ├── conformance_burntsushi_valid.cpp
│   ├── conformance_iarna_invalid.cpp
│   ├── conformance_iarna_valid.cpp
│   ├── cpp.hint
│   ├── for_each.cpp
│   ├── formatters.cpp
│   ├── impl_toml.cpp
│   ├── leakproof.hpp
│   ├── lib_catch2.hpp
│   ├── main.cpp
│   ├── manipulating_arrays.cpp
│   ├── manipulating_parse_result.cpp
│   ├── manipulating_tables.cpp
│   ├── manipulating_values.cpp
│   ├── meson.build
│   ├── odr_test_1.cpp
│   ├── odr_test_2.cpp
│   ├── parsing_arrays.cpp
│   ├── parsing_booleans.cpp
│   ├── parsing_comments.cpp
│   ├── parsing_dates_and_times.cpp
│   ├── parsing_floats.cpp
│   ├── parsing_integers.cpp
│   ├── parsing_key_value_pairs.cpp
│   ├── parsing_spec_example.cpp
│   ├── parsing_strings.cpp
│   ├── parsing_tables.cpp
│   ├── path.cpp
│   ├── settings.hpp
│   ├── tests.cpp
│   ├── tests.hpp
│   ├── user_feedback.cpp
│   ├── using_iterators.cpp
│   ├── visit.cpp
├── vs
│   ├── odr_test.vcxproj
│   ├── test_debug_x64_cpplatest_noexcept_unrel.vcxproj
│   ├── test_debug_x64_cpplatest_noexcept.vcxproj
│   ├── test_debug_x64_cpplatest_unrel.vcxproj
│   ├── test_debug_x64_cpplatest.vcxproj
│   ├── test_debug_x64_noexcept_unrel.vcxproj
│   ├── test_debug_x64_noexcept.vcxproj
│   ├── test_debug_x64_unrel.vcxproj
│   ├── test_debug_x64.vcxproj
│   ├── test_debug_x86_cpplatest_noexcept_unrel.vcxproj
│   ├── test_debug_x86_cpplatest_noexcept.vcxproj
│   ├── test_debug_x86_cpplatest_unrel.vcxproj
│   └── test_debug_x86_cpplatest.vcxproj

```

```

├── test_debug_x86_noexcept_unrel.vcxproj
├── test_debug_x86_noexcept.vcxproj
├── test_debug_x86_unrel.vcxproj
├── test_debug_x86.vcxproj
├── test_release_x64_cpplatest_noexcept_unrel.vcxproj
├── test_release_x64_cpplatest_noexcept.vcxproj
├── test_release_x64_cpplatest_unrel.vcxproj
├── test_release_x64_cpplatest.vcxproj
├── test_release_x64_noexcept_unrel.vcxproj
├── test_release_x64_noexcept.vcxproj
├── test_release_x64_unrel.vcxproj
├── test_release_x64.vcxproj
├── test_release_x86_cpplatest_noexcept_unrel.vcxproj
├── test_release_x86_cpplatest_noexcept.vcxproj
├── test_release_x86_cpplatest_unrel.vcxproj
├── test_release_x86_cpplatest.vcxproj
├── test_release_x86_noexcept_unrel.vcxproj
├── test_release_x86_noexcept.vcxproj
├── test_release_x86_unrel.vcxproj
├── test_release_x86.vcxproj
├── windows_compat.cpp
├── toml++.code-workspace
├── toml.hpp
├── toml++.natvis
├── toml++.props
├── toml++.sln
├── toml-test
├──   ├── meson.build
├──   ├── README.md
├──   ├── tt_decoder.cpp
├──   ├── tt_decoder.vcxproj
├──   ├── tt_encoder.cpp
├──   ├── tt_encoder.vcxproj
├──   └── tt.hpp
├── toml++.vcxproj
├── toml++.vcxproj.filters
├── tools
├──   ├── ci_single_header_check.py
├──   ├── clang_format.bat
├──   ├── generate_conformance_tests.py
├──   ├── generate_single_header.bat
├──   ├── generate_single_header.py
├──   ├── generate_windows_test_targets.py
├──   ├── requirements.txt
├──   ├── utils.py
├──   └── version.py
├── vendor
├──   ├── catch.hpp
├──   ├── json.hpp
├──   └── README.md
├── tomlplusplus.wrap
├── websocketpp.wrap
├── zlib.wrap
├── zstd.wrap
├── tci-bridge
├──   └── tci-rigctld.mjs      # TCI-Server für Hamlib-Funkgeräte (IC-7300...), läuft beim Hörer –
siehe docs/RIG_CONTROL.md
├── thermal_guard.py          # CPU-Überhitzungsschutz für das Admin-Panel (läuft auch
eigenständig)
├── thermal-guard.service    # Beispiel-systemd-Unit für den Wächter, für Installationen ohne
Admin-Panel
├── tmpfiles
├──   └── phantomsdr-logs.conf # hält admin.log + proxy.log im Besitz des Panel-Benutzers (Pfade
vor der Installation anpassen)
├── update.sh

```

```

└─ waterfall.sh
└─ websdr_relay.json.example # Konfigurationsvorlage (Port, Limits, Stationskennung)
└─ websdr_relay.py          # das Relay selbst – siehe docs/RECEIVE_DIVERSITY.md
└─ websocketpp_asio_connection.hpp # Boost-≥-1.87-Shim: ws_post statt io_service::post
└─ websocketpp_asio_endpoint.hpp  # Boost-≥-1.87-Shim: ws_work / ws_restart,
max_listen_connections
└─ websocketpp_asio.hpp          # Boost-≥-1.87-Shim für websocketpp (io_context,
executor_work_guard)
└─ xgo.sh                       # alt: startet spectrumserver, von check-go.sh aufgerufen

```

Wurzelverzeichnis

Konfigurationsdateien

Datei	Zweck	Wann zu ändern
config.toml	Standardkonfiguration	Ersteinrichtung, Tests
config-rtl.toml	RTL-SDR-spezifische Konfiguration	Bei Verwendung eines RTL-SDR
config-rspla.toml	Konfiguration für SDRplay RSP1A	Bei Verwendung eines RSP1A
config-airspyhf.toml	Konfiguration für Airspy HF+	Bei Verwendung eines Airspy
config-fobos-hf.toml	RigExpert Fobos SDR, HF1/HF2 Direktabtastung (0-25 MHz)	Bei Verwendung eines Fobos auf KW
config-fobos.toml	RigExpert Fobos SDR, RF-Zweig (25-6000 MHz)	Bei Verwendung eines Fobos oberhalb 25 MHz
config-hackrf.toml	Konfiguration für HackRF One	Bei Verwendung eines HackRF
config-rx888mk2.toml	Konfiguration für RX888 MK2	Bei Verwendung eines RX888
config.example.hackrf.toml	Beispiel für HackRF One	Bei Verwendung eines HackRF

Start-/Stopp- und Wartungsskripte

Jedes `start-*.sh` unten ist ein **eigenständiger Starter + Watchdog + Protokollierer**: Es beendet eine laufende Instanz, startet Empfänger + `spectrumserver`, löst sich in den Hintergrund ab, startet die Kette bei einem Absturz automatisch neu und protokolliert in `logwebsdr.txt`. Sie teilen sich ein Stopp-Skript und eine einzige `flock`-Sperre (es läuft immer nur ein Empfänger). Ändern Sie nur den Block **RECEIVER CONFIGURATION** am Anfang (Empfängerargumente / Konfiguration / Prozessname). Die sechs Startskripte aus derselben Vorlage (`start-airspyhf.sh`, `start-rtl.sh`, `start-rspla.sh`, `start-fobos.sh`, `start-fobos-hf.sh`, `start-hackrf.sh`) sind unterhalb dieses Blocks identisch; `start-rx888mk2.sh` ist ein eigenes Skript. Seit v5.0.0 können mehrere Empfänger nebeneinander laufen: Ein Startskript mit `INSTANCE=<name>` gestartet, bekommt der Empfänger seinen eigenen Ordner in `instances/<name>/` sowie eigenes Log, eigene Sperre und eigenen FIFO — siehe [Mehrere Empfänger](#).

Skript	Zweck
<code>install.sh</code>	Automatisierte Installation und Erstellung
<code>add-receiver.sh</code>	Fügt diesem Rechner einen weiteren Empfänger hinzu: fragt, welcher Empfänger und was er abdeckt, installiert seinen Treiber, legt <code>instances/<name>/</code> und den Eintrag in <code>receivers.toml</code> an und bietet an, ihn zu starten und den Proxy neu zu starten. Die Installer bieten es am Ende an — siehe Mehrere Empfänger
<code>start-rtl.sh</code>	Start + Watchdog des Servers mit RTL-SDR (<code>rtl_sdr</code>)
<code>start-rsp1a.sh</code>	Start + Watchdog des Servers mit SDRplay RSP1A (<code>rx_sdr</code>); verwendet libmirisdr-5 oder die SDRplay-API, wenn diese installiert ist (<code>RX_DRIVER</code> erzwingt einen)
<code>start-airspyhf.sh</code>	Start + Watchdog des Servers mit Airspy HF+ (<code>rx_sdr</code>)
<code>start-fobos-hf.sh</code>	Start + Watchdog des Servers mit RigExpert Fobos SDR, HF1/HF2 Direktabtastung (<code>rx_sdr \ cf32_to_real</code>)
<code>start-fobos.sh</code>	Start + Watchdog des Servers mit RigExpert Fobos SDR, RF-Zweig (<code>rx_sdr</code>)
<code>start-hackrf.sh</code>	Start + Watchdog des Servers mit HackRF One (<code>hackrf_transfer</code>)
<code>start-rx888mk2.sh</code>	Start + Watchdog des Servers mit RX888 MK2 (<code>rx888_stream</code>)
<code>start-all.sh</code>	Startet jeden in <code>receivers.toml</code> aufgeführten Empfänger über sein eigenes Startskript — auf einer Station mit mehreren Empfängern als Startskript des Admin-Panels eintragen; siehe Mehrere Empfänger
<code>stop-websdr.sh</code>	Beendet Server + Watchdog — von allen Empfängern gemeinsam genutzt
<code>stop-websdr.sh</code> <code><instanz></code>	Stoppt nur einen Empfänger (<code>main</code> für den Hauptempfänger); ohne Argument stoppen wie bisher alle Empfänger
<code>setup-rx888-udev.sh</code>	Installiert udev-Regeln, damit <code>rx888_stream</code> des RX-888 ohne sudo läuft
<code>setup-rsp1a.sh</code> / <code>setup-fobos.sh</code> / <code>setup-airspyhf.sh</code>	Installieren die SoapySDR-Treiberkette für RSP1A, Fobos oder Airspy HF+ — SoapySDR aus der Distribution, Treiber und <code>rx_tools</code> gebaut in <code>sdr_drivers/</code> , eine udev-Regel — auf jeder der vier unterstützten Distributionen. Die Installer-Optionen 3, 5 und 6 rufen sie auf; sie laufen auch allein. Gemeinsamer Code in <code>setup-sdr-common.sh</code> — siehe Installationsanleitung
<code>setup-hackrf.sh</code>	Installiert das Paket <code>hackrf</code> der Distribution und eine udev-Regel für den HackRF One, auf jeder der vier unterstützten Distributionen; die Installer-Option 7 ruft es auf
<code>setup-rtlsdr.sh</code>	Installiert den RTL-SDR-Treiber auf jeder der vier unterstützten Distributionen — das Paket <code>rtl-sdr</code> der Distribution oder mit <code>RTL_V4=y</code> den aus dem Quelltext gebauten Treiber für den RTL-SDR Blog V4 —, sperrt den DVB-T-Treiber und legt eine udev-Regel an; <code>add-receiver.sh</code> ruft es auf
<code>setup-firewall.sh</code>	Optionaler Flutschutz auf Kernel-Ebene: lädt eine nftables-Tabelle mit einer Obergrenze gleichzeitiger Verbindungen und einer Rate je Quelladresse auf den Empfängerports, einer SSH-Bruteforce-Bremse und Windows-Dateifreigabe, die außerhalb privater Bereiche geschlossen wird. Braucht root, kann Sie nicht aussperren (policy accept, bestehende Verbindungen werden zuerst angenommen) und <code>--apply</code> nimmt sich selbst zurück, wenn nicht binnen 60 s bestätigt — siehe Verbindungslimits
<code>setup-cpufreq-perms.sh</code>	Gibt einer Gruppe <code>cpufreq</code> Schreibzugriff auf das CPU-Frequenzlimit, damit die Throttle-Stufe des Wächters ohne root funktioniert. Installiert eine <code>tmpfiles.d</code> -Regel, damit es einen Neustart übersteht; <code>--revoke</code> macht es rückgängig
<code>update.sh</code>	Die Installation vom veröffentlichten Stand aktualisieren, ohne Konfiguration, Marker, Frequenzliste und eigene Änderungen anzurühren — siehe Installationsanleitung
<code>recompile.sh</code>	Backend und/oder Frontend neu bauen und die unter <code>/</code> ausgelieferte Variante wählen

Skript	Zweck
<code>smeter_theme.sh</code>	Standard-Skalenbild des analogen S-Meters (dark / amber / vintage) für alle Nutzer setzen und den Frontend-Neubau anbieten — siehe Varianten bearbeiten
<code>waterfall.sh</code>	Ändert die voreingestellte minimale Wasserfall-Ebene (dB) in <code>waterfall.js</code> + <code>App.svelte</code> — siehe README
<code>kiwi_install.sh</code>	Installiert die KiwiSDR-Client-Emulation in einen Baum, der sie noch nicht hat: patcht die Backend-Quellen, kopiert <code>src/kiwi_bridge.h</code> und fügt einen dokumentierten <code>[kiwi_emulation]</code> -Block in die Konfigurationsdateien im Wurzelverzeichnis ein. Idempotent, und sichert jede Datei, die es anfasst — siehe Emulation von KiwiSDR-Clients
<code>tci-bridge/tci-rigctld.mjs</code>	Wird vom Empfänger nicht benutzt. Ein kleines Node.js-Programm, das ein Hörer neben seinem eigenen Transceiver ausführt: Es liest Frequenz, Betriebsart und Sendezustand aus Hamlibs <code>rigctld</code> und stellt sie auf Port 50001 als TCI bereit, sodass die Schaltfläche TCI-CAT der Seite ein Funkgerät ohne eigenes TCI steuern kann, etwa den IC-7300 — siehe Transceiver-Steuerung

Alte Startkette. `go.sh`, `xgo.sh`, `check-go.sh`, `kill.sh` und `_relaunch.sh` sind die vorige Generation von Start-, Watchdog- und Stoppskripten. Alles, was sie taten, steckt heute in jedem `start-<radio>.sh`, und das ist das, was Sie benutzen sollten. Sie liegen noch auf der Platte, weil bestehende Installationen sie referenzieren, und werden nicht mehr gepflegt.

Datendateien

Datei	Zweck	Format
<code>markers.json</code>	Frequenz-Lesezeichen und -Marker	JSON
<code>chat_history.txt</code>	Chatnachrichten der Nutzer	Klartext
<code>favicon.ico</code>	Website-Symbol	ICO-Bild
<code>fftw_wisdom</code>	FFT-Optimierungsdaten	FFTW-Binärdatei
<code>phantom_fftw_wisdom</code>	Zusätzliche FFT-Optimierung	FFTW-Binärdatei

Admin-Panel und Spot-Reporting

Datei	Zweck
<code>admin_server.py</code>	Das Admin-Panel selbst. Neben den Verwaltungsseiten betreibt es den Sampler der Seite Graphen : ein Hintergrund-Thread erfasst alle 2 Sekunden CPU-Takt, Last, Temperatur und Nutzer online und hält sie nur im Speicher — 1 Stunde in voller Auflösung plus 24 Stunden als 30-Sekunden-Mittelwerte. Es wird nichts auf die Festplatte geschrieben, der Verlauf geht beim Neustart verloren.
<code>admin_config.json</code>	Einstellungen des Admin-Panels (Passwort-Hash, Ports, Optionen, Schwellen des thermischen Schutzes)
<code>thermal_guard.py</code>	CPU-Überhitzungsschutz. Stoppt den Server, wenn die CPU zu heiß wird, und startet ihn wieder, sobald sie abgekühlt ist; die Schwellen leitet er aus dem kritischen Grenzwert ab, den Ihre eigene CPU veröffentlicht, statt aus einer festen Zahl. Er versucht nie zu erkennen, was den Server überwacht — solange es zu heiß ist, wiederholt er den Stopp alle 2 Sekunden, sodass ein Watchdog, eine systemd-Unit oder ein Cron-Job, der ihn wiederbelebt, rückgängig gemacht wird, bis die Maschine abkühlt. Nur Standardbibliothek; wird von <code>admin_server.py</code> importiert (getaktet vom Graphen-Sampler) und läuft für Installationen ohne Panel auch eigenständig. Standardmäßig nur Protokollmodus, handelt also erst nach Aktivierung — siehe [Admin-Panel]siehe das Thermal-Guard-Handbuch
<code>autorun/</code>	Der Spot-Reporting-Daemon — siehe Autorun Spot Reporter
<code>autorun.json</code>	Zu dekodierende Bänder/Betriebsarten, Identität und Ziele
<code>autorun-status.json</code>	Laufender Zustand des Daemons — speist die Zähler des aktuellen Laufs (die Kacheln je Decoder), die bei Stop/Start zurückgesetzt werden
<code>autorun-totals.json</code>	Gesamtwerte seit Beginn je Band und Betriebsart — die Zahl neben jedem Kontrollkästchen; vom Daemon geschrieben, damit sie Neustarts überstehen

Dateien des Build-Systems

Datei	Zweck
<code>meson.build</code>	Hauptkonfiguration des Builds
<code>meson_options.txt</code>	Konfigurierbare Build-Optionen
<code>.gitattributes</code>	Attribute des Git-Repositories

Quellcode (`src/`)

Das Verzeichnis `src/` enthält die C++-Implementierung des Backends.

Wichtige Komponenten

1. Hauptanwendung (`main.cpp`)

- Wertet die Kommandozeilenargumente aus
- Lädt die Konfigurationsdatei
- Initialisiert die Serverkomponenten
- Startet die Ereignisschleife

2. Spektrumserver (`spectrumserver.cpp`)

- Koordiniert alle Komponenten
- Verwaltet die Nutzerverbindungen

- Verteilt die Spektrumdaten
- Bearbeitet Nutzeranfragen

3. SDR-Treiber (drivers/)

- Abstrakte Schnittstelle für SDR-Hardware
- Liest und formatiert Abtastdaten
- Behandelt gerätespezifische Funktionen

4. DSP-Engine (dsp/)

- FFT-Berechnung (CPU-/GPU-beschleunigt)
- Demodulation (AM, FM, SSB, CW usw.)
- Audiofilterung und Resampling
- AGC und Rauschminderung

5. Webserver (server/)

- WebSocket-Kommunikation
- Ausliefern statischer HTTP-Dateien
- Verwaltung der Nutzersitzungen
- Echtzeit-Datenstreaming

6. Audiokodierung (audio/)

- FLAC-Kompression
- Opus-Kompression
- Optimierung des Streamings

Frontend (frontend/)

Die mit Svelte und Vite erstellte webbasierte Benutzeroberfläche.

1. Hauptanwendung (App.svelte)

- Oberste Komponente
- Layoutstruktur
- Orchestrierung der Komponenten
- **Decoder-Tastenreihe** — eine Taste je Decoder im Hauptpanel, direkt unter der Modus-Auswahl; ein Druck startet den Decoder und öffnet sein Fenster, ein erneuter Druck stoppt ihn. Sie hat die frühere Bandbreiten-Reihe ersetzt. RADEL/RADEU liegen stattdessen in `lib/ModesSelector.svelte`, neben der Modus-Auswahl und in den Pop-ups **Modes** und **Bands**.

2. Wasserfallanzeige (waterfall.js + lib/)

- Canvas-basierte Darstellung von Spektrum und Wasserfall, Farbpaletten und die adaptive Auto-Anpassung, alles in `waterfall.js` (reines JS, keine Komponente)
- Interaktive Abstimmung über `lib/PassbandTuner.svelte`
- Bandplan- und Marker-Overlays über `lib/FrequencyMarkers.svelte`
- Das Audio-Spektrogramm ist eine eigene Komponente: `lib/Spectrogram.svelte`

3. Bedienelemente (App.svelte + lib/)

- Frequenzeingabe/-anzeige — `lib/FrequencyInput.svelte`

- Betriebsartenwahl (AM/FM/SSB/CW) — `lib/ModesSelector.svelte`, Bandumschaltung — `lib/BandSelector.svelte`
- AGC/NR/NB und die übrigen Bedienelemente liegen in `App.svelte` selbst; eine eigene `Controls.svelte` gibt es nicht

3a. Scanner (`scanner.js`)

- Kanalsuchlauf: führt den Empfänger über einen Bereich und hält beim ersten Kanal mit Signal an. Reines JS, keine Komponente — `App.svelte` liefert VFO, Betriebsart, Bandplan und den Abstimmufruf und erhält den UI-Zustand über einen einzigen Callback zurück
- Die Schwelle ist ein Abstand in dB zum Grundrauschen des Bandes statt eines absoluten Pegels und nutzt das Rauschen, das `waterfall.js` ohnehin nachführt (`snrNoiseDb`), braucht also keine eigene Kalibrierung
- Zwei Arten, das Band zu durchlaufen: jeden Kanal abstimmen und hineinhören, oder zuerst das Spektrum prüfen und nur abstimmen, was sich nicht ausschließen lässt. Das Spektrum darf einen Kanal überspringen, aber nie auf einem anhalten — jeder Halt stammt aus einem echten Verweilen
- Der Bereich ist entweder das Band aus dem Bandplan oder genau der sichtbare Wasserfall; Fortsetzen nach Ruhe, die Höchstverweildauer und die Ausschlussliste liegen ebenfalls hier und werden in `localStorage` gesichert

4. Audiosystem (`audio.js`)

- WebSocket-Audiostrom
- FLAC-/Opus-Dekodierung
- Steuerung der Audiowiedergabe
- Verteilt rohes PCM (vor AGC, Rauschminderung und Stummschaltung abgegriffen) an die Betriebsartendecoder
- KI-Rauschunterdrückung: `lib/rnnoise.js` lädt das RNNoise-WebAssembly-Modul (`@jitsi/rnnoise-wasm`) beim ersten Gebrauch; `audio.js` wendet es nur in Sprachbetriebsarten an, hinter dem Abgriff für die Decoder, sodass diese es nie hören
- FLAC läuft über `flacLowLatency.js`, das jedes Paket in dem Moment decodiert, in dem es eintrifft, und die tatsächliche Rate des Streams (zum Beispiel 12016,3 Hz) mit einem 64-Tap-Resampler (2,7 ms Verzögerung) auf 12000 Hz bringt. Der WASM-Decoder in `modules/` gab den Ton in festen Blöcken von 1024 Samples frei und hielt rund 160 ms zurück; er wird weiterhin für Opus, für Diversity-Quellen und für FLAC verwendet, wenn die Seite mit `?flac=wasm` geöffnet wird
- Steuerung des Wiedergabepuffers (`playoutControl.js`): `ArrivalMeter` misst, wie unregelmäßig die Pakete gemessen an der eigenen Zeitachse des Streams eintreffen, und die Drift zwischen dem Abtasttakt des Empfängers und der Soundkarte des Hörers; `CushionController` hält den kleinsten Pufferstand der letzten Minute nahe an der Reserve der gewählten Pufferstufe, indem er eine tonhöhengerechte Periode (`spliceOut` / `spliceIn`) des Tons zum Lautsprecher herausschneidet oder wiederholt. Decoder und Aufnahmen erhalten den unveränderten Stream. Gilt für den Pfad mit geplanten Puffern, den jeder Hörer einer reinen http-Station nutzt; `?drain=0` schaltet ihn ab, `?diag=1` zeigt seine Werte

4a. Betriebsartendecoder und ihre Worker

Jeder der rechenintensiven Betriebsartendecoder läuft in einem eigenen Web Worker, sodass die Dekodierung niemals die Audiowiedergabe oder den Wasserfall blockiert. Sie folgen einem gemeinsamen Muster — drei Dateien je Decoder:

Decoder	Engine	Worker	Proxy im Haupt-Thread
SSTV	sstv.js	sstv.worker.js	sstvWorkerProxy.js
HF-FAX	fax.js	fax.worker.js	faxWorkerProxy.js
NAVTEX + FSK/RTTY + PSK31 + Olivia	fsk.js, psk31.js, olivia.js	fsk.worker.js	fskWorkerProxy.js
CW	cwDecoder.js	cw.worker.js	cwWorkerProxy.js

- Die **Engine** ist reiner DSP-Code ohne Kenntnis von Workern und lässt sich daher auch direkt ausführen (Unit-Tests oder der Rückfall in den Haupt-Thread).
- Der **Worker** hält eine Engine-Instanz und leitet deren Ereignisse unverändert weiter.
- Der **Proxy** spiegelt die Methodenoberfläche der Engine, sodass `audio.js` ihn genau so aufruft, wie es den Decoder aufrufen würde. Er erzeugt den Worker verzögert beim ersten Einschalten und weicht auf die Ausführung im Haupt-Thread aus, wenn keine Worker verfügbar sind.

Zwei Details sind tragend: Das PCM wird in einen frischen Puffer **kopiert**, bevor es an den Worker übertragen wird (das Übertragen einer Sicht auf den Audio-Akkumulator würde diesen ablösen und die Wiedergabe beenden), und die `init`-Nachricht des Workers wendet die Konfiguration erneut auf eine bereits laufende Engine an, statt eine neue vorzusetzen.

`fsk.js` bedient sowohl NAVTEX als auch FSK/RTTY aus einer Engine, je Instanz über ein `role`-Feld gewählt; jede Instanz besitzt ihren eigenen Zustand, sodass beide unabhängig laufen können.

Die Rolle `fsk` beherbergt zusätzlich zwei Decoder, die überhaupt kein FSK sind. Wird die Variante `psk31` oder `olivia` gewählt, übergibt `fsk.js` das Audio an `psk31.js` bzw. `olivia.js` statt an die eigene Diskriminator-Kette, nutzt aber weiterhin dessen Konfiguration, Worker und Ereignis-Infrastruktur — `fsk.worker.js`, `fskWorkerProxy.js` und `audio.js` müssen von beiden Betriebsarten nichts wissen, und die Oberfläche verarbeitet durchgehend dieselben `char/status/metrics`-Ereignisse.

- `psk31.js` — BPSK31: komplexes Basisband, angepasstes Filter, Differenzdemodulation und Varicode, mit spektraler Grobsuche und einer Feinregelung über etwa ± 25 Hz.
- `olivia.js` — Olivia MFSK: eine Portierung des MFSK-Empfängers von Pawel Jalochoa aus `fldigi` (`pj_mfsk.h`, GPL-3, wie auch dieses Projekt), einschließlich der Walsh/Hadamard-Fehlerkorrektur und der blinden Synchronisationssuche über Blockphase und Frequenzversatz.
- `broadcastSchedules.js` — die UTC-Sendepläne, die FAX-, NAVTEX- und RTTY-Decoder als Voreinstellungen anbieten, aus den NOAA/NWS-Marine-Faxplänen und den veröffentlichten NAVTEX-Stationslisten

4b. Empfangsdiversität (`diversity.js`)

- Verbindet den lokalen Empfänger mit einem zweiten anderswo und folgt dem Standort, der gerade das bessere Signal hat. Reines JS, keine Komponente — es sitzt an einer Nahtstelle in `audio.js`, die ihm das lokale PCM übergibt und wiedergibt, was zurückkommt
- **Auswahl, nicht Addition.** Zwei Standorte hören dieselbe Aussendung über verschiedene ionosphärische Wege, ihre Signalverläufe haben also unabhängige Phase; addiert klingt das kammgefiltert. Kohärentes Zusammenführen bräuchte einen gemeinsamen Takt, den zwei Empfänger über das Internet nicht teilen. Gemischt wird nur während einer 30-ms-Überblendung
- Die Ausrichtung korreliert die beiden **Audio-Hüllkurven** (logarithmische Leistung bei 100 Hz), nie die Signalverläufe — die Hüllkurve übersteht sowohl den Weg als auch jeden Codec. Eine Ausrichtung gilt erst, wenn eine zweite, unabhängige Suche zustimmt; das weist die überzeugte, aber falsche Verzögerung zurück, die zwei gegenphasig schwindende Standorte sonst liefern

- Der entfernte Strom wird zuerst auf die lokale Rate **eingerastet**. Zwei Empfänger sind zwei Takte und zwei Dezimierungsketten, ihr Audio trifft also bis zu 2% auseinander ein, selbst wenn beide 12 kHz melden — 240 Abtastwerte pro Sekunde Drift, die keine Korrelation hält. Das Verhältnis wird daraus gemessen, wie viele Abtastwerte jede Seite tatsächlich liefert, und von einem Resampler angewandt, der seine gebrochene Phase über Blockgrenzen mitführt, sodass ein beliebiges Verhältnis dauerhaft hält
- Die Standortwahl nutzt ein Perzentil-SNR, gemessen an **inhaltlich ausgerichteten** Abtastwerten, mit Hysterese, Haltezeit und einem schnellen Notausgang, wenn der aktive Standort zusammenbricht. Die Pegel werden Rauschen-zu-Rauschen angeglichen, damit ein Wechsel das Grundrauschen nicht verändert
- Die Decoder behalten den **lokalen** Datenstrom: FT8, JS8, WSPR und RADE integrieren kohärent über einen Zeitschlitz, und ein Wechsel mittendrin ist eine Phasenunstetigkeit, die die Dekodierung kosten kann

4c. Diversitätsquellen (`remoteSource.js`, `kiwiSource.js`, `uberSource.js`, `webSdrSource.js`)

- Ein Vertrag — `onPcm` / `onState` / `tune` / `canReceive` — sodass `diversity.js` nie erfährt, was am anderen Ende hängt. Ein weiterer Empfängertyp ist eine neue Datei
- `remoteSource.js` — ein weiterer PhantomSDR-Plus über `/audio` (cbor + FLAC), nutzt `createDecoder()` aus `lib/wrappers.js`
- `kiwiSource.js` — ein KiwiSDR: SND-Rahmen, Big-Endian-PCM, samt dem 10-Byte-GPS-Zeitstempel, den ein Stereopakete vor das Audio setzt
- `uberSource.js` — ein UbersDR über sein eigenes `/ws`: Opus in einem 21-Byte-Kopf, Abstimmen über die offene Verbindung. Die Sitzungs-ID muss zuvor per `POST /connection` angemeldet werden und muss eine UUID sein
- `webSdrSource.js` — ein WebSDR, über `websdr_relay.py` auf diesem Server: der Browser kann sich nicht direkt verbinden, weil WebSDR den `Origin`-Header prüft und kein Skript ihn ändern darf. Abgestimmt wird per Textrahmen auf derselben Verbindung; die Bandabdeckung liefert das Relay
- `webSdrCodec.js` — das WebSDR-Audioformat: ein byteweise getaggter Strom, dessen komprimierte Blöcke einen 20-stufigen Leaky-LMS-Prädiktor speisen. Aus dem WebSDR-Client portiert und Sample für Sample dagegen geprüft
- `diversityList.js` — die Liste gespeicherter Empfänger und die Adressregeln der vier Typen, gemeinsam genutzt von der Desktop-Oberfläche und der Mobilseite, damit ein Format beiden dient. Dazu die Übertragung — ein kompakter Blob, sein QR-Code und ein Parser, der jede Form annimmt, die die beiden Seiten je geschrieben haben — sowie `browseUrl()`, das eine angewählte Adresse in eine zurückverwandelt, die ein Browser öffnen kann
- `lib/DiversityPanel.svelte` — die Oberfläche: Adresse, Quellentyp, SNR-Trim und Live-Status, dazu die gespeicherten Empfänger — benannt, frei sortierbar und als JSON-Datei exportier- und importierbar, je Quellentyp im `localStorage`. Bearbeitet wird alles innerhalb des Feldes: `prompt()` und `confirm()` blockieren den Main-Thread, über den die Audioblöcke in das Playback-Worklet geschoben werden. Eine Schaltfläche  **QR** zeichnet die Liste als scannbaren Code, denn `localStorage` gehört zu einem Browser und ein Telefon beginnt leer. `mobile/Mobile.svelte` bietet dieselbe Funktion in einem Reiter **Div**, im eigenen einfachen CSS jener Seite. Siehe [Empfangsdiversität](#)

4d. Modus-Erkennung (`modeId.js`, `modePriors.js`)

- Beantwortet „was höre ich da?“. Sie liest denselben rohen PCM-Abgriff wie die Decoder und listet die wahrscheinlichen Betriebsarten nach Rang, damit man den richtigen Decoder wählt, statt alle zehn durchzuprobieren. Sie decodiert nie: sie misst physikalische Eigenschaften des Signals und bewertet sie gegen eine Tabelle bekannter Betriebsarten

- Die belegte Bandbreite ist die zusammenhängende –15-dB-Breite um die Spitze, bewusst kein 99%-Leistungswert: Tastklicks legen lange Ausläufer auf das Leistungsintegral und ließen jede schmale Betriebsart um ein Vielfaches zu breit erscheinen. Die Symbolrate kommt aus der **Momentanfrequenz** statt aus Tonenergien, weil keine Integrationszeit zugleich einen 170-Hz-Hub und ein 100-Bd-Symbol auflösen kann. Die Tastrate kommt aus der Ein/Aus-Hüllkurve und dient zugleich als CW-Tempoanzeige
- `modePriors.js` bringt den einen Hinweis ein, den das Audio nicht tragen kann: wo Sie abgestimmt sind. Ein 100-Bd-/170-Hz-Signal auf 518 kHz ist NAVTEX; dasselbe Signal auf 14,070 MHz ist es nicht. Es gewichtet nur um, was das Signal ohnehin hergab, und erfindet nie einen Kandidaten
- FT8, JS8 und FT2 werden absichtlich als eine Gruppe gemeldet: an Bandbreite und Tonabstand allein sind sie nicht zu trennen, und etwas anderes zu behaupten wäre eine selbstbewusst falsche Antwort
- Unterhalb von etwa 10 dB SNR schweigt sie, statt zu raten
- Läuft in einem eigenen Web Worker (`modeId.worker.js` + `modeIdWorkerProxy.js`) nach demselben Engine/Worker/Proxy-Muster wie die Decoder oben; das Ergebnis ist der Chip in `lib/ModeIdChip.svelte`

4e. Abgewiesene Verbindungen (`refused.js`, `clientVersion.js`)

- `refused.js` ist das gemeinsame Vokabular für eine Verbindung, die der Server abweist: Schließcode **4003** (über einem Limit pro Adresse, oder die Seite ist älter als `[server] min_client_version`) und **4001** (ein Sysop-Rauswurf). Beide sind endgültig. `audio.js`, `waterfall.js` und `events.js` importieren alle `isRefusal()` von dort, denn alle drei öffnen eine Verbindung, die abgewiesen werden kann, und alle drei müssen dann ihr Init-Promise auflösen — sonst wartet die Seite ewig auf eine Verbindung, die nie kommt
- Nichts versucht es erneut. Eine abgebrochene `/audio`-Verbindung beendet die Sitzung mit Absicht: `/waterfall` und `/events` kamen mit einem Wiederverbinden nie zurück, eine erneut aufgebaute Sitzung war also lebendes Audio an einem eingefrorenen Wasserfall, und gegen ein Ratenlimit würde jeder Versuch genau die Abweisung verlängern, die er umgehen wollte. Die Desktop-Seite erklärt eine Abweisung beim Laden und hört einfach auf, wenn eine mitten in der Sitzung kommt; `/mobile` zeigt eine Zeile, die zum Neuladen auffordert
- `clientVersion.js` enthält eine einzige Ganzzahl, `CLIENT_VERSION`, die die Seite ihrer Audioverbindung als `/audio?v=N` anhängt. Der Server weist alles unterhalb von `[server] min_client_version` ab — so zwingt eine Station Tabs, die noch einen älteren Stand fahren, zum Neuladen, der einzige Hebel, den es gibt, da der Server JavaScript im Browser nicht mehr erreicht. Erhöhen Sie sie, wenn eine Frontend-Änderung von offenen Tabs nicht weiter ignoriert werden darf
- Vollständige Referenz: [Verbindungslimits](#)

4f. Chat-Antworten (`lib/chatReply.js`)

- Eine Antwort ist eine gewöhnliche Chat-Nachricht, die mit einer Markierung beginnt, welche die beantwortete Nachricht benennt: `[RE:<Zeitstempel>|<Name>]`. Der Server speichert und verteilt sie wie jede andere Zeile und braucht keine Änderung
- `chatReply.js` erzeugt und liest diese Markierung und ordnet Antworten eine Ebene tief unter ihrem Original an. `App.svelte` und `mobile/Mobile.svelte` verwenden es beide, sodass beide Seiten den Chat gleich gliedern
- Ist das Original bereits aus dem Verlauf von 20 Nachrichten des Servers verschwunden, bleibt die Antwort dort, wo sie eintraf, und zeigt nur Name und Uhrzeit dessen, worauf sie antwortet

5. Zustandsverwaltung (`stores/`)

- Reaktive Datenspeicher
- Gemeinsamer Anwendungszustand
- Ereignisbehandlung

Wichtige Funktionen

- Audioverarbeitung in Echtzeit
- Dekodierung digitaler Betriebsarten (FT8, RTTY usw.)
- Audiofilterung
- Spektrumanalyse

Frequenzlisten (`frequencylist/`)

Die Frequenzmarken auf dem Wasserfall. `mymarkers.json` ist die Liste, die der Empfänger tatsächlich anzeigt; der Rest ist das Rohmaterial, das `update-markers.sh` daraus macht, aufgefrischt aus den Online-Sendeplänen. `README.md` in diesem Verzeichnis erklärt die Aktualisierung in allen sieben Sprachen.

```
frequencylist/
├─ mymarkers.json           # die Marken, die der Empfänger zeigt – von Hand gepflegt, übersteht
eine Aktualisierung
├─ shortwavestations.json   # Kurzwellensender, erzeugt von generate-current-shortwave.py
├─ 0.TXT                   # der globale A26-KW-Sendeplan, von update-markers.sh entpackt und
ausgewertet
├─ a26allx2.zip            # Quellarchiv, von update-markers.sh heruntergeladen
├─ admin.txt · antenna.txt · broadcas.txt · fmorg.txt · language.txt · site.txt
├─                          # Hilfslisten aus derselben Quelle (Standorte, Sprachen, Betreiber,
Antennen)
├─ generate-current-shortwave.py
├─ update-markers.sh       # frischt alles Obige aus den Online-Sendeplänen auf
└─ README.md
```

Format (`mymarkers.json` , `shortwavestations.json`)

```
[
  { "frequency": 77500,   "name": "DCF77", "mode": "CW" },
  { "frequency": 2485000, "name": "Vanuatu Broadcasting", "mode": "AM" }
]
```

Konfigurationsdateien

Serverkonfiguration (`.toml` -Dateien)

Aufbau der Konfigurationsdateien:

```

[server]
# Web server settings
port = 9002
html_root = "frontend/dist/"
threads = 2
otherusers = 1

[limits]
# Limits pro Adresse – alle standardmäßig aus oder auf einem sinnvollen Wert,
# eine Konfiguration ohne sie verhält sich wie bisher. Siehe CONNECTION_LIMITS.md.
per_ip = 3          # gleichzeitige Hörer von einer Adresse
per_ip_rate = 40     # neue Verbindungen pro Minute von einer Adresse

[websdr]
# Online registration
register_online = true
name = "WebSDR Name"
antenna = "Antenna Type"
grid_locator = "AB12cd"
hostname = "domain.com"

[input]
# SDR input settings
sps = 2048000        # Sample rate
fft_size = 131072    # FFT size
frequency = 145000000 # Base frequency
signal = "iq"         # Signal type: "iq" or "real"
audio_sps = 12000     # Audio sample rate
audio_compression = "opus" # "flac" or "opus"
accelerator = "opencl" # "none", "cuda", "opencl"

[input.driver]
# Driver settings
name = "stdin"
format = "u8"          # Sample format

[input.defaults]
# User interface defaults
frequency = 145500000
modulation = "FM"

```

Standortinformationen (site_information.json)

```

{
  "siteSysop": "Operator Callsign",
  "siteSysopEmailAddress": "email@example.com",
  "siteGridSquare": "AB12cd",
  "siteCity": "City, Country",
  "siteInformation": "https://github.com/sv1btl/PhantomSDR-Plus",
  "siteHardware": "Hardware specs",
  "siteSoftware": "Software version",
  "siteReceiver": "SDR model",
  "siteAntenna": "Antenna description",
  "siteNote": "Additional notes",
  "siteIP": "http://domain.com:9002",
  "siteSDRBaseFrequency": 0,
  "siteSDRBandwidth": 2048000,
  "siteRegion": 1,
  "siteChatEnabled": true
}

```

Build-System

Meson-Build-Konfiguration

meson.build (Wurzel)

Legt fest:

- Projektmetadaten
- Abhängigkeiten
- Compileroptionen
- Listen der Quelldateien
- Build-Ziele

meson_options.txt

Verfügbare Optionen:

```
option('opencl', type: 'boolean', value: false, description: 'Enable OpenCL support')
option('cuda', type: 'boolean', value: false, description: 'Enable CUDA support')
option('optimization', type: 'string', value: '3', description: 'Optimization level')
```

Dateiabhängigkeiten

Build-Abhängigkeiten des Backends

```
spectrumserver binary depends on:
├─ C++ source files (src/**/*.cpp)
├─ External libraries:
│   ├─ FFTW3
│   ├─ WebSocket++
│   ├─ FLAC
│   ├─ Opus
│   ├─ Liquid-DSP
│   ├─ Boost
│   ├─ zlib
│   ├─ zstd
│   └─ OpenCL/CUDA (optional)
└─ Subproject headers:
    ├─ nlohmann/json
    └─ toml11
```

Build-Abhängigkeiten des Frontends

```
frontend/dist/ depends on:
├─ Source files (frontend/src/**/*.*)
├─ npm packages (node_modules/):
│   ├─ Svelte
│   ├─ Vite
│   └─ @wasm-audio-decoders/opus-ml
│   └─ ...
└─ Static assets (frontend/public/)
```

Datenfluss

Ablauf des Serverbetriebs

1. SDR Hardware → rtl_sdr/hackrf_transfer/etc.
↓
2. Sample Stream → stdin → spectrumserver
↓
3. spectrumserver:
 - FFT calculation (waterfall)
 - Demodulation (audio)
 - Compression (FLAC/Opus)
 ↓
4. WebSocket → Browser Client
↓
5. Browser:
 - Render waterfall
 - Decode and play audio
 - Display controls

Ablauf der Nutzerinteraktion

1. User clicks on waterfall
↓
2. JavaScript sends frequency change request
↓
3. WebSocket → spectrumserver
↓
4. spectrumserver:
 - Updates demodulator frequency
 - Sends new audio stream
 ↓
5. Browser receives and plays new audio

Leitfaden zum Ändern von Dateien

Wenn Sie Backend-Code ändern (src/**):

```
cd PhantomSDR-Plus
meson compile -C build
# Server restart required
```

Wenn Sie Frontend-Code ändern (frontend/src/**):

```
cd PhantomSDR-Plus/frontend
npm run build
cd ..
# Server restart required (for static files)
```

Wenn Sie die Konfiguration ändern (.toml, .json):

```
# Restart server
./stop-websdr.sh
./start-rtl.sh # (or appropriate start script)
```

Wenn Sie die Marker ändern (`markers.json`):

```
# Reload page in browser
# No server restart needed
```

Wichtige Pfade

Laufzeitpfade

- **Konfiguration:** `./config-*.toml`
- **HTML-Wurzel:** `./frontend/dist/`
- **Marker:** `./markers.json`
- **Chatverlauf:** `./chat_history.txt`
- **FFTW-Wisdom:** `./fftw_wisdom`, `./phantom_fftw_wisdom`

Build-Pfade

- **Binärausgabe:** `./build/spectrumserver`
- **Frontend-Ausgabe:** `./frontend/dist/`
- **Node-Module:** `./frontend/node_modules/`

Quellpfade

- **Backend-Quellen:** `./src/`
- **Frontend-Quellen:** `./frontend/src/`
- **DSP-Bibliotheken:** `./jsdsp/`

Übliche Dateioperationen

Eine neue SDR-Konfiguration hinzufügen

1. Vorhandene Konfiguration kopieren: `cp config-rtl.toml config-mydevice.toml`
2. Parameter bearbeiten: `nano config-mydevice.toml`
3. Startskript erstellen: `cp start-rtl.sh start-mydevice.sh`
4. Startskript bearbeiten: `nano start-mydevice.sh` — ändern Sie nur den Block **RECEIVER CONFIGURATION** am Anfang (`RX_LABEL`, `RX_COMM` = der Prozessname des Empfängers, `RX_ARGS`, `CONFIG`, `FIFO` sowie der `prestart`-Hook, falls das Gerät einen benötigt). Die darunterliegende Logik für Start/Watchdog/Protokollierung ist allgemein und muss nicht geändert werden.
5. Ausführbar machen: `chmod +x start-mydevice.sh`

`stop-websdr.sh` beendet bereits jedes `start-*.sh --watchdog`; verwendet Ihr Empfänger einen anderen Prozessnamen als `rx888_stream/rx_sdr/rtl_sdr/hackrf_transfer/cf32_to_real`, ergänzen Sie dort zusätzlich eine Zeile `killall -9 <Name>`.

Das Frontend anpassen

1. Quelle ändern: `nano frontend/src/App.svelte`
2. Neu bauen: `cd frontend && npm run build && cd ..`
3. Server neu starten: `./stop-websdr.sh && ./start-rtl.sh`

Eigene Marker hinzufügen

1. Markerdatei bearbeiten: `nano markers.json`
 2. Format:

```
json { "markers": [ { "frequency": 145500000, "label": "2m Calling", "mode": "FM" } ] }
```
 3. Browser neu laden (kein Serverneustart nötig)
-

Versionsverwaltung

In Git zu verfolgende Dateien

- Quellcode (`src/`, `frontend/src/`, `jsdsp/`)
- Konfigurationsbeispiele (`config.example*.toml`)
- Build-System (`meson.build`, `meson_options.txt`)
- Dokumentation (`*.md`, `docs/`)
- Skripte (`*.sh`)

Zu ignorierende Dateien (`.gitignore`)

- Build-Ergebnisse (`build/`, `frontend/dist/`)
 - Abhängigkeiten (`frontend/node_modules/`)
 - Nutzerdaten (`chat_history.txt`)
 - Persönliche Konfigurationen (`config-rtl.toml`, sofern angepasst)
 - Binärdaten (`*.o`, `*.so`)
-

Diese Strukturdokumentation soll Ihnen helfen, sich in der Codebasis von PhantomSDR-Plus zurechtzufinden und sie zu verstehen.

Einrichtungsanweisungen finden Sie in [INSTALLATION.md](#). Hinweise zur Bedienung finden Sie in [USER_GUIDE.md](#).

73 de SV1BTL & SV2AMK