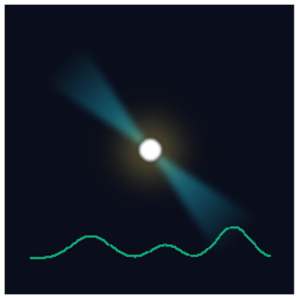
DSES Radio Astronomy Workbench

Installation Guide



Version 1.8.0

Richard M Hambly (K0GD)

DSES

November 2026

Contents

1. What the recipient needs to know first 4
System requirements 5
2. Installing Radioconda 5
2.1 Windows 11 5
2.2 Linux 6
2.3 macOS (Intel and Apple Silicon) 7
2.4 Add the packages the Workbench needs (all platforms) 7
3. Installing the DSES Radio Astronomy Workbench 8
3.1 Windows 11 8
3.2 Linux 9
3.3 macOS 10
3A. Extra setup for SDRPlay receivers (RSP1A / RSP1B / RSPduo / RSPdx) 11
3A.1 Install the SDRplay API 11
3A.2 Install the SoapySDRPlay module 11
4. First-run checklist 13
Recommended one-time VOLK tuning 13
5. Updating to a new version 14
Auto-update notifications 14
Performing the update — in-app (easiest) 14
Performing the update — manually 15
What is preserved across versions 15
Backing up your settings 15
6. Troubleshooting 16
Windows: nothing happens when I start it 16
“Radioconda not found in standard locations” 16
“Missing required packages (PySide6, pyqtgraph, and/or scipy)” — or ModuleNotFoundError: No module named 'PySide6' 16
“No UHD Devices Found” 17
Persistent “O” overflows at high sample rates 17
Spectrum looks flat at −140 dB 18
macOS: “developer cannot be verified” 18
macOS: harmless startup messages in the terminal 18
Settings won’t persist 18
7. Appendix A — Settings file reference 18
8. Appendix B — Device selection and SigMF playback 19
Choosing among multiple radios 19
SigMF playback mode (no-device fallback) 20
9. Appendix C — Operating Guide 20

Starting up — device selection 21
Playback mode 21
Control panels (dockable) 21
Spectrum (top plot) 30
Waterfall (bottom plot) 31
Persistence 32
Update checks 32
Tips for pulsar work 32

DSES Radio Astronomy Workbench — Installation Guide

Version 1.8.0 Author: Richard M Hambly (K0GD) — k0gd@cnssys.com License: GPL-3.0-or-later

This document covers:

- How to install the DSES Radio Astronomy Workbench on Windows 11, Linux, and macOS (Intel and Apple Silicon).
- How to update to a new version.
- Common troubleshooting and a reference for advanced settings.

The application itself is one Python file plus a few launchers and an icon. The heavy machinery — GNU Radio, UHD, SoapySDR, PySide6, NumPy, SciPy — is supplied by **Radioconda**, which you install separately as a one-time prerequisite.

Renamed in 1.4.0. Through version 1.3.4 this program was the *DSES Spectrum Analyzer*. That name stopped describing it once it grew pulsar observation planning, science-format recording, PRESTO folding, multi-day HI drift-scan campaigns, and a built-in pulsar simulator, so from 1.4.0 it is the **DSES Radio Astronomy Workbench** (the Workbench for short). Existing installs update in place exactly as before — see *Coming from 1.3.4 or earlier* in the update section for the few visible differences.

Supported radios. The program is designed around the Ettus USRP B210 but also drives any SoapySDR-compatible receiver: SDRPlay RSP1A / RSP1B / RSPduo / RSPdx (with one extra setup step — see §3A), RTL-SDR, HackRF, Airspy / Airspy HF+, BladeRF, LimeSDR, and PlutoSDR. The sample-rate combo and gain slider adapt automatically to whichever device you pick at startup.

1. What the recipient needs to know first

Each recipient performs **two installs**, in this order (plus a third optional step if you're using SDRPlay):

1. **Install Radioconda** (one-time). Radioconda is a curated conda distribution that bundles GNU Radio, UHD, SoapySDR, and the surrounding scientific Python stack for software-defined radio. Use a current build — releases are date-stamped (e.g. `2025.03.14`) and any recent one ships UHD ≥ 4.8, which the B210 needs. Download from the releases page:

- **Primary:** <https://github.com/radioconda/radioconda-installer/releases>
- **Backup:** <https://github.com/ryanvolz/radioconda/releases> (redirects to the primary)

   Per-OS file names and steps are in §2 below. Then add three packages the app needs that Radioconda doesn't bundle — `pyside6`, `pyqtgraph`, `scipy` — with one `conda install` (§2.4). (§2)

2. **Install this app**, which is just a small zip of Python and launcher scripts. (§3)

3. **If using an SDRPlay receiver:** after the two installs above, do the extra SDRPlay setup in §3A (install the SDRplay API + the SoapySDRPlay module). It comes last because it copies a file from the app's zip into Radioconda's folders. Other supported radios need no extra setup.

Updates to the app afterwards are a small zip replace — Radioconda does not need to be re-installed for every release.

System requirements

Component	Minimum
OS	Windows 11; Ubuntu/Debian/Fedora/Arch (current); macOS 11 Big Sur or newer
CPU	x86_64 *or* Apple Silicon (M1/M2/M3/M4)
RAM	4 GB free
USB	One USB 3.0 port (5 Gbps) for the B210
Hardware	Ettus USRP B210 *or* any SoapySDR-supported receiver (SDRPlay RSPx, RTL-SDR, HackRF, Airspy, BladeRF, LimeSDR, PlutoSDR). With one radio attached the app uses it silently; with several, a picker appears. See §8 for details.
Disk	~3 GB for Radioconda, plus ~1 MB for this app
Display	1280 × 800 minimum; 14″ MacBook Pro is the layout target

2. Installing Radioconda

2.1 Windows 11

1. Download the Windows installer — the asset named **`radioconda-<version>-Windows-x86_64.exe`** (e.g. `radioconda-2025.03.14-Windows-x86_64.exe`) — from the Radioconda releases page. Pick the newest release and grab the `…-Windows-x86_64.exe` file.

- **Primary:** <https://github.com/radioconda/radioconda-installer/releases>
- **Backup:** <https://github.com/ryanvolz/radioconda/releases> (redirects to the primary)

2. Double-click the installer. Accept defaults; the installer offers to put Radioconda under `C:\ProgramData\radioconda` (system-wide) or `%LOCALAPPDATA%\radioconda` (per-user). Either works; the app's launcher checks both.
3. If you have a radio, plug it into a USB 3 port. A B210 silently installs its UHD driver from Radioconda the first time it's connected (no prompt).
4. (Optional) Confirm your radio is detected — open the **Anaconda Prompt (Radioconda)** shortcut from the Start menu and run the command for your device:

- **B210 / USRP:** `uhd_find_devices` — lists the unit with its own serial number.
- **RTL-SDR, HackRF, Airspy, BladeRF, LimeSDR, PlutoSDR:** `SoapySDRUtil --find` — lists the device under its driver name.
- **SDRPlay (RSPx):** do §3A first, then `SoapySDRUtil --find` shows it as `driver = sdrplay`.
- **No radio yet:** skip this — the app starts in playback mode using the bundled sample file (see §8), which is ideal for training.

2.2 Linux

1. Download the Linux installer — the asset named **`radioconda-<version>-Linux-x86_64.sh`** (e.g. `radioconda-2025.03.14-Linux-x86_64.sh`) — from the releases page:

- **Primary:** <https://github.com/radioconda/radioconda-installer/releases>
- **Backup:** <https://github.com/ryanvolz/radioconda/releases> (redirects to the primary)

   (ARM and POWER builds — `…-Linux-aarch64.sh`, `…-Linux-ppc64le.sh` — are published too if you're on one of those.)

2. From a terminal, run the file you downloaded (substitute the real version):

```
bash radioconda-<version>-Linux-x86_64.sh
```

   Accept the license and let it install to `~/radioconda` (the launcher checks this path automatically).

3. **USB permissions (B210 / USRP only).** If you have a B210, activate the Ettus udev rules so the device is reachable without root:

```
sudo cp ~/radioconda/lib/uhd/utils/uhd-usrp.rules /etc/udev/rules.d/

sudo udevadm control --reload-rules && sudo udevadm trigger
```

   Without this you'll get permission errors when the B210 is plugged in. Then add yourself to the `usrp` (or `plugdev`) group if your distro uses one, and log out/in.

   Other USB SDRs (RTL-SDR, HackRF, Airspy, etc.) have their own udev rules, usually installed with the device's own package. SDRPlay uses its API service (see §3A) and needs no udev rules.

4. (Optional) Confirm your radio is detected, using the command for your device:

```
~/radioconda/bin/uhd_find_devices      # B210 / USRP

~/radioconda/bin/SoapySDRUtil --find   # RTL-SDR, HackRF, Airspy, SDRPlay (after §3A), …
```

No radio yet? Skip this — the app starts in playback mode from the bundled sample (see §8).

2.3 macOS (Intel and Apple Silicon)

Download from the releases page:

- **Primary:** <https://github.com/radioconda/radioconda-installer/releases>
- **Backup:** <https://github.com/ryanvolz/radioconda/releases> (redirects to the primary)

Radioconda publishes builds for both Mac architectures — pick the one matching your CPU (`uname -m` prints `x86_64` or `arm64`):

- **Intel:** `radioconda-<version>-MacOSX-x86_64.sh` (e.g. `radioconda-2025.03.14-MacOSX-x86_64.sh`)
- **Apple Silicon (M1/M2/M3/M4):** `radioconda-<version>-MacOSX-arm64.sh`

(Each Mac build also comes as a double-click `.pkg` graphical installer — `…-MacOSX-x86_64.pkg` / `…-MacOSX-arm64.pkg` — if you prefer that over the shell script.)

1. From a terminal, run the file you downloaded (substitute the real version + arch):

```
bash radioconda-<version>-MacOSX-<arch>.sh
```

2. Accept the default install location (`~/radioconda`).
3. If you have a radio, plug it in. macOS needs no driver install for the B210.
4. (Optional) Confirm your radio is detected, using the command for your device:

```
~/radioconda/bin/uhd_find_devices      # B210 / USRP

~/radioconda/bin/SoapySDRUtil --find   # RTL-SDR, HackRF, Airspy, SDRPlay (after §3A), …
```

No radio yet? Skip this — the app starts in playback mode from the bundled sample (see §8).

5. **Quarantine note.** If you ever see a Gatekeeper "cannot verify developer" dialog on a Radioconda binary, run:

```
xattr -dr com.apple.quarantine ~/radioconda
```

2.4 Add the packages the Workbench needs (all platforms)

Stock Radioconda bundles GNU Radio, UHD, and SoapySDR, but **not** the GUI and plotting packages this app uses (`pyside6`, `pyqtgraph`, `scipy`). Install them once into Radioconda — this is a one-time step; future app updates don't repeat it.

Open an **activated Radioconda shell**:

- **Windows:** the **Anaconda Prompt (Radioconda)** shortcut in the Start menu.
- **Linux / macOS:** any terminal whose prompt shows `(base)` (or run `source ~/radioconda/bin/activate`).

Then run:

```
conda install -c conda-forge pyside6 pyqtgraph scipy
```

Accept the prompt; conda downloads and installs the three packages. If the launcher later reports "Missing required packages," this is the step that was skipped.

3. Installing the DSES Radio Astronomy Workbench

Download the application from the distribution site. You can either use the direct links below, or browse the folder <https://gpstime.com/sw_distribution/dses-workbench/> and pick the newest `dses-workbench-*.zip`:

- **Application (zip):** <https://gpstime.com/sw_distribution/dses-workbench/dses-workbench-1.8.0.zip>
- **This guide (PDF):** <https://gpstime.com/sw_distribution/dses-workbench/DSES_Radio_Astronomy_Workbench_Installation.pdf>

The zip is roughly 60 MB — it includes a short sample recording so the program can run in playback mode when no radio is attached. (Replace the version number in the link if a later release has been published.)

Give the program folder a name without the version number. The zip unpacks into a folder named for its version (`dses-workbench-1.8.0`). Rename that folder — `DSES_Workbench` on Windows, `dses-workbench` on Linux and macOS in the steps below — before you create a shortcut. Updates from inside the program replace the files in place and the folder keeps its name, so a folder named for a version would be wrong after the first update.

3.1 Windows 11

1. Extract the zip into a **permanent folder** you have write permission for, for example `C:\DSES`. Do not run the program out of Downloads: each new download lands beside the old copies under a slightly different name, and Windows' own cleanup (Storage Sense) can be set to empty that folder. (Versions through 1.6.0 could not start from a folder whose full path contained a space; 1.8.0 fixed that.) The extracted folder is named for the version, `dses-workbench-1.8.0\`: **rename it** to `DSES_Workbench`, so that the program lives in `C:\DSES\DSES_Workbench\`. It contains `dses_workbench.py`, `launcher.bat`, `launcher.ps1`, `install-shortcut.ps1`, `LICENSE`, the `icons\` folder, and this guide.
2. Double-click **`launcher.bat`** to start the application.

3. The first time you run it, the launcher searches for Radioconda in this order: the `RADIOCONDA_ROOT` environment variable, any currently-activated conda env, `%LOCALAPPDATA%\radioconda`, `C:\ProgramData\radioconda`, `%USERPROFILE%\radioconda`, a cached config file, then `conda info --base` if `conda` is on PATH. If none of these find a working install, you'll get a prompt asking for the path; type it in and the launcher remembers it for next time.
4. (Recommended) Create a desktop shortcut (with the app's pulsar icon) by running **`install-shortcut.ps1`**. The program never creates this icon by itself: extracting the zip does not, and an in-app update keeps whatever shortcut you already have. The reliable way — which works regardless of your PowerShell execution policy — is to open PowerShell in the extracted folder and run:

```
powershell -NoProfile -ExecutionPolicy Bypass -File .\install-shortcut.ps1
```

   It prints `Created: …\DSES Radio Astronomy Workbench.lnk`. (Right-clicking the script → "Run with PowerShell" also works *if* your machine's execution policy allows local scripts; if nothing appears, the policy blocked it — use the command above instead.) The shortcut is named **DSES Radio Astronomy Workbench** and launches the app via `launcher.ps1`. It stores the folder's location, so run this step again if you ever rename or move the program folder.

After the first run, the app's window opens with the spectrum and waterfall plots. Tuning, sample-rate, gain, and recording controls are in the sidebar on the right.

3.2 Linux

1. Extract the zip somewhere under your home directory where you have write permission — a conventional spot for a per-user app is `~/Applications`. For example:

```
mkdir -p ~/Applications && cd ~/Applications

unzip ~/Downloads/dses-workbench-1.8.0.zip

mv dses-workbench-1.8.0 dses-workbench

cd dses-workbench

chmod +x launcher.sh
```

   The `mv` line gives the folder a name without the version. The `dses-workbench/` folder contains `dses_workbench.py`, `launcher.sh`, `dses-workbench.desktop`, `LICENSE`, the `icons/` folder, and this guide. (Avoid system locations like `/opt` unless you extract with `sudo` — keeping it in your home directory avoids permission issues.)

2. Launch:

```
./launcher.sh
```

3. (Optional) Install a desktop entry so the app appears in your application menu:

```
INSTALL_DIR="$(pwd)"

sed "s|__INSTALL_DIR__|$INSTALL_DIR|g" dses-workbench.desktop \

    > ~/.local/share/applications/dses-workbench.desktop

update-desktop-database ~/.local/share/applications/ 2>/dev/null || true
```

(The `.desktop` file uses `__INSTALL_DIR__` as a placeholder so the launch path is correct on whichever machine installs it. The `sed` line substitutes the current directory into the placeholder.)

3.3 macOS

The steps are the same as Linux. Extract it into your personal **`~/Applications`** folder (Finder shows it as your own Applications folder, separate from the system `/Applications`):

```
mkdir -p ~/Applications && cd ~/Applications

unzip ~/Downloads/dses-workbench-1.8.0.zip

mv dses-workbench-1.8.0 dses-workbench

cd dses-workbench

chmod +x launcher.sh

./launcher.sh
```

Keeping it under your home directory (rather than the system `/Applications`) avoids permission prompts and Gatekeeper friction.

(Optional) Create a Desktop icon. Run **`install-shortcut.command`** to build a double-clickable **DSES Radio Astronomy Workbench.app** on your Desktop, using the app's pulsar icon. Launching it starts the Workbench with no Terminal window. In Finder, right-click `install-shortcut.command` → **Open** (the first run may need Gatekeeper approval — see below), or from a terminal:

```
bash install-shortcut.command
```

It prints `Created: …/Desktop/DSES Radio Astronomy Workbench.app`. Re-run it any time you move the extracted folder (the app remembers the location it was built from). The first time you double-click the new `.app`, macOS may warn about an app from an unidentified developer — right-click it → **Open** once, or

approve it under **System Settings** → **Privacy & Security** → **Open Anyway**. Startup logs go to `~/Library/Logs/DSES_Workbench.log` if you ever need to troubleshoot a launch.

On Apple Silicon, **make sure** you installed the `arm64` build of Radioconda. Mixing an `x86_64` Radioconda with a native `arm64` Python or Qt will produce confusing errors at startup. There is no Rosetta-only step required if both halves match the CPU.

If the Finder warns about an unidentified developer when running `launcher.sh`, clear the quarantine attribute on the unzipped folder:

```
xattr -dr com.apple.quarantine dses-workbench
```

3A. Extra setup for SDRPlay receivers (RSP1A / RSP1B / RSPduo / RSPdx)

Do this only if you're using an SDRPlay receiver — and only *after* you've installed Radioconda (§2) and extracted this app (§3), because the steps below put files into Radioconda's folders and use a file that ships inside the app's zip. B210, RTL-SDR, HackRF, Airspy, BladeRF, LimeSDR, and PlutoSDR users can skip this section entirely.

SDRPlay needs two extra pieces: the manufacturer's API driver, and the SoapySDRPlay module that bridges that driver to the SoapySDR layer the program uses.

3A.1 Install the SDRplay API

Download and install it from <https://www.sdrplay.com/api/> — pick the installer for your OS.

- **Windows:** run the `.exe` installer; accept defaults. A reboot may be needed.
- **Linux:** run the `.run` installer with `sudo`.
- **macOS:** run the `.pkg` installer.

After installation, verify the API can see your unit with the SDRplay Service / Status app (Windows) or `SDRplayService` (Linux/macOS). You should see your RSP listed.

3A.2 Install the SoapySDRPlay module

This module is **not** available through `conda install` on any platform, so the steps differ by OS.

Windows — copy the pre-built module that ships in this distribution:

The extracted release folder (from §3) contains `sdrplay\sdrPlaySupport.dll`. Copy it into Radioconda's SoapySDR module directory. That directory is under `C:\ProgramData`, so you need an **Administrator** PowerShell (right-click Windows PowerShell → "Run as administrator"):

```
Copy-Item ".\sdrplay\sdrPlaySupport.dll" `
  "C:\ProgramData\radioconda\Library\lib\SoapySDR\modules0.8\" -Force
```

(Run it from the extracted release folder, or give the full path to the DLL.) If your Radioconda is installed somewhere else, adjust the path — the target is always `…\Library\lib\SoapySDR\modules0.8\`.

Then start the SDRplay API service (also from the Administrator PowerShell — the installer leaves it stopped):

```
Set-Service SDRplayAPIService -StartupType Automatic
Start-Service SDRplayAPIService
```

Linux — install from your package manager:

```
Debian / Ubuntu:   sudo apt install soapysdr-module-sdrplay
Fedora:            sudo dnf install SoapySDRPlay
```

macOS — install from the Pothosware Homebrew tap:

```
brew tap pothosware/homebrew-pothos
brew install soapysdrplay3
```

Verify it loaded (any OS, from a Radioconda prompt):

```
SoapySDRUtil --info
```

…should list `sdrplay` in the "Available factories" line. If it doesn't:

- Re-check that the SDRplay API from §3A.1 installed correctly.
- On Windows, confirm the DLL landed in `modules0.8\` and the `SDRplayAPIService` is running (`Get-Service SDRplayAPIService`).

When you launch the Workbench with an RSP attached, it appears in the device picker as e.g. `RSP1B — 240513BE60  [sdrplay]`.

4. First-run checklist

Regardless of OS, before declaring the install good:

- The window title bar reads `DSES Radio Astronomy Workbench — v1.1.6 — <radio model> — <serial>` (e.g. `… — USRP B210 — 3273A91` or `… — RSP1B — 240513BE60`). The version number must match the bundle you installed; the radio portion confirms which device is being used.
- Pull up **Help → User Guide** from the menu bar. The guide should open.
- Pull up **Help → About**. The author, version, license, and the path to the settings file should be readable.
- The spectrum plot should show live data (not a flat line at −140 dB). If it's flat, the radio isn't streaming — see Troubleshooting §6.
- Click the **408 MHz** preset under Pulsar Band. The center frequency should retune and the trace should redraw.

Recommended one-time VOLK tuning

GNU Radio's number-crunching uses **VOLK**, which can profile your CPU once and pick the fastest SIMD kernels (AVX2, NEON, etc.) for the rest of your machine's life. Without the profile, you'll see this warning every time you launch the app:

```
[WARNING] SoapyVOLKConverters: no VOLK config file found.

          Run volk_profile for best performance.
```

To run it once and silence the warning:

```
Windows: open the "Anaconda Prompt (Radioconda)" shortcut, then:

  volk_profile


Linux / macOS: open a terminal with Radioconda on PATH, then:

  volk_profile
```

Takes about 30 seconds. Writes the chosen kernels to `%APPDATA%\.volk\volk_config` on Windows or `~/.volk/volk_config` on Linux/macOS. The warning disappears next time you launch the analyzer, and FFT throughput improves on machines with newer SIMD instruction sets.

5. Updating to a new version

Auto-update notifications

The program checks for new versions at launch — silently, no more than once per 24 hours; nothing appears if you're already up to date. When a newer version *is* available, a dialog opens with the release notes and these choices:

- **Install Update…** — download and install it from inside the app (see *In-app update* below).
- **Read the Guide (PDF)** — opens this guide first (recommended before installing).
- **Download .zip** — just downloads the bundle so you can apply it by hand.
- **Skip this version** (won't nag again about that specific version) / **Remind me later**.

You can also trigger a check at any time via **Help → Check for Updates…**. The automatic check never installs anything on its own — an update is only applied when you click **Install Update…**. To disable the auto-check entirely, set `auto_check = false` under `[updates]` in your `settings.ini`.

Performing the update — in-app (easiest)

Click **Install Update…** in the update dialog. The app downloads the new bundle, **verifies its published checksum** before changing anything, and asks how to install it:

- **Update this installation** (default) — replaces the current version in place. The files it overwrites are backed up first, so a failed update rolls back instead of leaving a broken install. When it finishes, it offers to **restart** into the new version.
- **Install a new copy** — installs into a folder you choose and **keeps** the current version. Leave **Add a desktop shortcut** ticked and it creates a *separate* icon labelled with the new version (e.g. *DSES Radio Astronomy Workbench 1.1.6*), so both versions stay launchable.

Radioconda does **not** need reinstalling for an app update.

Coming from 1.3.4 or earlier (the rename release). The 1.4.0 update installs over your existing folder like any other. What changes: the program file is now `dses_workbench.py` (the old `dses_spectrum_analyzer.py` is left behind as a tiny stand-in that just starts the new file, so anything still pointing at it keeps working); the release folder on the download server moved from `b210_sa` to `dses-workbench`, and the update URL stored in your settings file is moved there automatically on first launch; and your existing desktop shortcut, macOS `.app`, or Linux menu entry keeps working because it runs the launcher, but it still carries the old name — re-run `install-shortcut.ps1` / `install-shortcut.command` (or the Linux `.desktop` step in §3.2) to get a *DSES Radio Astronomy Workbench* icon, then delete the old one. Your settings, recordings folder, and cached Radioconda path are untouched.

Performing the update — manually

If you'd rather apply it yourself, each release is a self-contained zip:

1. **Close** the running app.
2. **Extract** the new zip to a temporary place and copy everything inside its `dses-workbench-<version>` folder over the files in your existing program folder, replacing them. The program folder keeps its name, so your shortcut keeps working.
3. **Run** it as usual.

What is preserved across versions

Your settings and window geometry are kept outside the app folder, so updates never wipe them. The on-disk locations are:

OS	Settings file
Windows	`%APPDATA%\DSES_Analyzer\settings.ini`
macOS	`~/Library/Application Support/DSES_Analyzer/settings.ini`
Linux	`~/.local/share/DSES_Analyzer/settings.ini`

The settings file is plain text and editable with any editor while the app is closed. To reset everything to defaults, either delete the file or use **Help → About → Restore Defaults…** in the running app.

Window size and position are saved in the same `settings.ini`, in the `[window]` section (plain `x` / `y` / `width` / `height` integers). You don't need to touch it; setting `width`/`height` to 0 (or deleting the lines) makes the window open at its default size next time.

Backing up your settings

The settings file is small (~1 KB). Copy it somewhere before a major upgrade if you want a quick rollback path:

```
Windows:  copy "%APPDATA%\DSES_Analyzer\settings.ini"
"%USERPROFILE%\Desktop\settings.ini.bak"

macOS:    cp "~/Library/Application Support/DSES_Analyzer/settings.ini"
~/Desktop/settings.ini.bak

Linux:    cp ~/.local/share/DSES_Analyzer/settings.ini ~/Desktop/settings.ini.bak
```

6. Troubleshooting

Windows: nothing happens when I start it

The launcher starts the program in a minimized console window, so a program that stops at once used to leave nothing on the screen. From 1.6.1 the launcher watches the first ten seconds: if the program stops in that time it shows a message with the exit code and the two lines to type to see the reason.

- **Version 1.6.0 or earlier in a folder whose full path contains a space** — a second download that the browser named `dses-workbench-1.6.0 (1)`, a `New folder`, a Windows user name with a space. The launcher handed Python a path cut off at the space, and nothing started. Install 1.6.1 or later, or move the program to a path without spaces.
- **To see the reason for any failed start**, open a Command Prompt (Start menu, type `cmd`) and run the program by hand from its folder: `cd /d "C:\DSES\DSES_Workbench"`, then `"C:\ProgramData\radioconda\python.exe" dses_workbench.py` (use your own program folder and Radioconda folder). The message it prints says what is wrong.
- **Keep the program out of Downloads** (see §3.1). One permanent folder plus the desktop shortcut avoids running an old or half-deleted copy by mistake.

"Radioconda not found in standard locations"

The launcher couldn't auto-detect a Radioconda install. Either:

- Install Radioconda to one of the standard paths the launcher checks (see §3.1 / §3.2 / §3.3), **or**
- Type the path when the launcher prompts you on first run, **or**
- Set the `RADIOCONDA_ROOT` environment variable to the install root before launching.

When you type the path, give the **install root** — the folder that contains `bin/python` (macOS/Linux) or `python.exe` (Windows), e.g. `~/radioconda` — **not** its `bin/` subfolder. A leading `~` is expanded to your home directory. On macOS/Linux, the launcher also accepts an already-activated conda environment, so if your shell prompt shows `(base)` for a Radioconda base env, just running the launcher from that shell is enough.

"Missing required packages (PySide6, pyqtgraph, and/or scipy)" — or ModuleNotFoundError: No module named 'PySide6'

Radioconda was found, but the app's GUI/plotting packages aren't installed in it. Do the one-time install from §2.4 — open an activated Radioconda shell and run:

```
conda install -c conda-forge pyside6 pyqtgraph scipy
```

Then launch again.

"No UHD Devices Found"

The OS doesn't see the B210. In order:

1. Unplug the B210 from USB. Wait 5 seconds. Plug it back in. UHD's USB endpoint can hang on some systems after an abrupt kill of the previous run.
2. Make sure you're using a USB 3 port (not USB 2 — the B210 will be flaky at high sample rates).
3. From a terminal:

```
Windows:  C:\ProgramData\radioconda\Library\bin\uhd_find_devices.exe

Linux:    ~/radioconda/bin/uhd_find_devices

macOS:    ~/radioconda/bin/uhd_find_devices
```

   If `uhd_find_devices` doesn't see it either, the problem is below our software — check the B210's LEDs and try a different USB cable.

4. If `uhd_find_devices` reports a **firmware/image error** rather than "no devices" — e.g. `Could not load firmware`, `ihex_reader::read(): No EOF record found`, or a missing FPGA image — your Radioconda's UHD images are incomplete or corrupt (seen on some macOS installs). Download them once with UHD's own tool, then re-check:

```
Windows:  C:\ProgramData\radioconda\Library\bin\uhd_images_downloader.exe

Linux:    ~/radioconda/bin/uhd_images_downloader

macOS:    ~/radioconda/bin/uhd_images_downloader
```

   It fetches ~100 MB of firmware/FPGA images into Radioconda (needs internet). After it finishes, `uhd_find_devices` should detect the B210.

Persistent "O" overflows at high sample rates

Some "O" characters in the Overflow box on the right sidebar at startup are normal as the USB pipe warms up. A continuous stream during steady-state operation usually means the host can't keep up with the sample rate:

- USB 2 ports cannot sustain 20 or 25 MS/s. Move to a USB 3 port.
- Other USB devices on the same controller (especially other 5 Gbps devices, or a heavily used external drive) compete for bandwidth. Move the B210 to its own controller if you have one.
- Reduce sample rate to 10 MHz or 16 MHz.

The Overflow box auto-clears after 15 seconds of no new overflows, so once the rate is stable you should see the box empty.

Spectrum looks flat at –140 dB

The GR flow graph is running but no samples are arriving. Usual causes:

- The B210 is in a powered-but-disconnected state. Unplug and replug.
- Antenna is disconnected (this app uses RX2 by default). Confirm a cable is on RX2.
- Sample rate is set to a value the B210 can't actually achieve. Try 20 MHz.

macOS: "developer cannot be verified"

```
xattr -dr com.apple.quarantine /path/to/dses-workbench
```

For the optional Desktop **`.app`** icon (built by `install-shortcut.command`), the same warning can appear the first time you double-click it — right-click the app → **Open** once, or approve it in **System Settings** → **Privacy & Security** → **Open Anyway**. You only need to do this once per machine.

macOS: harmless startup messages in the terminal

When launched from a terminal on macOS, the app prints a few benign lines that are **not** errors and can be ignored:

- `objc[...]: Class QT_ROOT_LEVEL_POOL... is implemented in both ...libQt6Core... and ...libQt5Core...` — Radioconda ships both Qt5 (for GNU Radio's own GUI blocks) and Qt6/PySide6 (what this app uses); both libraries are present in the process. The app forces the PySide6 backend, so this is cosmetic.
- `[ERROR] SoapySDR::loadModule(...) dlopen() failed` — SoapySDR probing a path that isn't a module. Harmless; the B210 is driven through UHD, not SoapySDR.

The app launches and runs normally despite these. (When started from the Desktop `.app`, they go to `~/Library/Logs/DSES_Workbench.log` instead of the screen.)

Settings won't persist

Settings should be saved automatically on every change and on app close. If they aren't:

- Check the path printed in **Help** → **About**. The file should exist and be writable.
- If the directory's parent doesn't exist, the app will fail to create it silently. On first run after install, click **Help** → **About** → **Open Settings Folder** to verify it opens cleanly.

7. Appendix A — Settings file reference

The settings INI is plain text and editable while the app is closed. Sections:

- `[tuning]` — preset/coarse/fine/manual frequency in Hz.
- `[rx]` — sample rate in Hz, RX gain in dB, `device_driver` (e.g. `uhd_b200`, `sdrplay`, `rtlsdr`), `device_serial`.

- `[recording]` — SigMF recording folder.
- `[spectrum]` — FFT size, window, averaging, max/min hold, Y-axis range, grid, axis-label toggles, dark/light background, trace styling for each background.
- `[waterfall]` — intensity range, colormap (per background), grid/axis-label toggles, row count.
- `[ui]` — control-panel visibility.
- `[window]` — saved window position/size as `x` / `y` / `width` / `height`. Set `width`/`height` to 0 (or delete the lines) to reset to the default window size.
- `[updates]` — `auto_check` (set to `false` to disable update notifications), plus internal book-keeping fields the program manages on its own.

Missing keys are filled in from built-in defaults on next launch. The file is rewritten on close with a header comment explaining what it is.

To reset every value to its default: use **Help → About → Restore Defaults…**, or just delete the file and launch the app.

8. Appendix B — Device selection and SigMF playback

Choosing among multiple radios

The program enumerates everything supported at launch — Ettus B210s via UHD, plus any SoapySDR-recognised receiver (SDRPlay, RTL-SDR, HackRF, Airspy, Airspy HF+, BladeRF, LimeSDR, PlutoSDR). Behavior:

- **No radio attached, no playback sample:** an error dialog says "No radio found" and explains how to enable playback (see below). The app exits.
- **No radio attached, but `sample.sigmf-data` + `sample.sigmf-meta` are present next to the program:** the app falls back to **SigMF playback mode** (see below). An informational dialog announces the fallback.
- **One radio attached:** the app opens it silently and remembers it in `settings.ini`.
- **Two or more radios attached** (any mix of B210s and SoapySDR receivers): a picker always appears so you choose which one to use. Your previously-used radio is pre-selected, so pressing Enter reuses the same one. The app remembers your selection for next launch.

The **RX** sidebar group has a button labeled `Device: <product> — <serial>` showing which radio the current session is using. Clicking it re-opens the picker; the new choice is saved and takes effect on the next launch.

The sample-rate combo and gain-slider range adapt automatically to whichever radio is open. SDRPlay caps at 10 MHz; RTL-SDR maxes around 3.2 MHz; HackRF goes to 20 MHz; the B210 to 25 MHz. Saved gain is clamped to the new device's range if you switch to a narrower one.

To pin a specific device permanently without using the picker, edit `settings.ini` and set `device_driver` + `device_serial` under `[rx]`. Use `device_serial = auto` to restore the default "first found" behavior.

SigMF playback mode (no-device fallback)

The distribution bundle includes a short SigMF recording so that users without any SDR attached can still launch the program, see live spectrum, and try the controls.

- Filenames: **`sample.sigmf-data`** (raw IQ) and **`sample.sigmf-meta`** (JSON metadata). Both live next to `dses_workbench.py` in your install folder.
- The file is read in a continuous loop, paced to match the original capture's sample rate.
- The window title shows **[Playback]**, the Device button shows the file name, and the Recording-status label reads `Playback (looping): sample.sigmf-data`.
- **Sample Rate**, **RX Gain**, and **Recording** controls are **disabled** — they have no meaning for a recorded file. Sample rate comes from the file's metadata.
- **Tuning IS enabled** in playback mode and works as a digital frequency shift on the file's baseband. Tuning to the file's actual center frequency (read from the .sigmf-meta) shows the recording's true content. Tuning to other frequencies within ±(sample_rate/2) of the file's center lets you explore the recorded bandwidth — useful for poking around inside the capture. Tune well outside that window and you'll see noise / wrap-around, since the recording doesn't contain data at those frequencies. This is the expected behavior for true I/Q data, not a bug.
- All visualization controls (Spectrum panel, Waterfall panel, including the dark/light background toggle, FFT averaging, max/min hold, intensity range, colormap, etc.) work normally.
- To replace the bundled sample with your own recording, do a real recording in live mode, then rename the resulting two files to `sample.sigmf-data` and `sample.sigmf-meta` and drop them next to the program. No source-code change is required.
- To skip playback mode and force an error exit when no SDR is attached, delete or rename either of the two `sample.sigmf-*` files.

9. Appendix C — Operating Guide

> This appendix is a copy of the program's built-in **Help → User Guide**. If the two ever differ, the in-app Help is authoritative for the version you're actually running (it ships inside the application).

The DSES Radio Astronomy Workbench (through version 1.3.4 the *DSES Spectrum Analyzer*) is one program for the club's SDR-based observing: a live spectrum analyzer and waterfall for the Ettus USRP B210 and other software-defined radios (SDRPlay RSP1A/RSP1B/RSPduo/RSPdx, RTL-SDR, HackRF, Airspy, BladeRF, LimeSDR, PlutoSDR via SoapySDR); a pulsar visibility planner; recording to SIGPROC filterbank, SigMF, and ezRA drift-scan formats; PRESTO folding and quick-look analysis; multi-day HI drift-scan

campaigns; and a B210 self test that doubles as a pulsar simulator. It began as an RFI survey tool and still serves for general spectrum monitoring.

Starting up — device selection

At launch the program enumerates attached SDRs (UHD + SoapySDR) and decides what to use:

- **One supported SDR attached**: opens it silently and remembers the driver and serial in the settings file.
- **Two or more radios attached**: a picker dialog always appears so you can choose which one to use. Your previous choice is pre-selected, so you can just press Enter to use the same radio as last time. Your selection is remembered for next launch.
- **No SDR attached, but a bundled SigMF sample is present**: the program falls back to **playback mode** — see below. It keeps watching for a receiver in the background: connect or power one on and, within a few seconds, a dialog offers a one-click restart to use it — no manual shut-down-and-relaunch dance.
- **No SDR and no sample**: an error dialog explains how to fix it and the program exits.

The window title always shows which radio is feeding the display, and the **RX** group has a `Device:` button you can click to re-open the picker. The new choice takes effect on the next launch.

Playback mode

If no SDR is attached, the program looks next to the application file for `sample.sigmf-data` + `sample.sigmf-meta` and, if both are found, plays the file back in a continuous loop. The window title shows [**Playback**]. The data source is the file, paced to the original capture's sample rate.

- **Sample Rate**, **RX Gain**, and **Recording** controls are disabled — they have no meaning for a recorded file. Sample rate comes from the file's metadata.
- **Tuning is enabled** and works as a digital frequency shift (`blocks.rotator_cc`) on the file's baseband. Tuning to the file's actual center frequency (read from the .sigmf-meta) shows the recording's true content; tuning to other frequencies within ±(sample_rate/2) of the file's center lets you "look around" inside the recorded bandwidth. Tune well outside that window and you'll just see noise / wrap-around — exactly what you'd expect, since the recording doesn't contain data at those frequencies.
- All visualization controls (Spectrum panel, Waterfall panel) work normally.
- To replace the sample with your own, save a SigMF recording, rename the two files to `sample.sigmf-data` and `sample.sigmf-meta`, and drop them next to the program. (You don't need to change any code.)

Control panels (dockable)

Every control panel is dockable. On the **right**: **Observation**, **Tuning**, **Radio**, and **Recording** — the science settings. On the **left**, beside the plots they belong to: **Spectrum Display** and **Waterfall Display**. Drag a panel by its title bar to rearrange, stack panels as tabs, tear one off into its own floating window (handy on a second

monitor), or close it; the **View** menu is organised by column — **Display Panels (left)** and **Control Panels (right)** — and each of those opens onto a *Show this column* switch that hides or restores the whole column in one click, followed by that column's individual panels. Hiding a column remembers which of its panels were open, so showing it again brings back exactly those (a panel you had closed on purpose stays closed). Your arrangement is remembered across runs. To float a panel, **drag it out by its title bar**. The title bar's three buttons act on it afterwards: the first **docks a floating panel back** into the window — which restores the default panel layout, the one arrangement Qt reliably rebuilds, so a panel can always be recovered. The second **enlarges** a floating panel just enough that all of its controls are visible; while enlarged the button shows a double-box *restore* icon and a click returns the panel to its previous size (resizing the panel by hand clears that state, so the next click enlarges afresh). It is greyed out while the panel is docked, where the layout sets the size. ✕ hides the panel. When a column runs out of room Qt stacks panels as tabs along its edge — those tabs are colored (pastel blue, DSES teal when selected) so the stack is easy to spot. The two display panels are deliberately restricted to the left column (they describe the plots, so they stay next to them) — they can still be reordered there, tabbed together, or floated freely. Control panels dock in either column but never along the top or bottom of the window, so a panel on its way to a floating spot is not snapped into a strip across the top. The menu bar (File / View / Radio / Recording / Help) duplicates the important actions, and long status messages — recording filenames, analysis progress — appear in the full-width **status bar** at the bottom of the window where they are never truncated.

Observe menu — Pulsars in View (Ctrl+P)

Answers "what can I record right now?" from the ATNF catalog: every pulsar above the site's elevation mask, sorted by flux *in the band you are tuned to* (S400 below ~900 MHz, S1400 above), with current az/el, period, DM, and how long each stays up. Selecting one fills the recording **Source** field and hands the recorder that pulsar's exact catalog RA/Dec for the `.fil` header — better than the position the app otherwise infers from the name.

- **Plan for a date and time**: the table normally shows the sky *now*. Tick **Plan for** — or just edit the date/time box, or use the `−1 d` / `−1 h` / `+1 h` / `+1 d` steps — and everything is recomputed for that instant instead: altitude, azimuth, time above the mask, next window, and which rows count as viable. Read the box as **UTC** (the convention in every file this app writes) or as your **Local** clock; switching keeps the same instant. Any date works, past as well as future, so you can also ask what was overhead when an old recording was made. The window title, the copied-table header line and the readout beside the box all say **PLANNED** so a planned table is never mistaken for the live sky, and **Now** puts it back. The readout also gives the site's **local sidereal time** — a source transits when LST equals its right ascension.

- **Hover for an explanation**: every column header explains what the column is, and *every individual cell* explains what its own value means — the delay this pulsar's DM produces across the band you are tuned to and inside one channel, when this source next crosses the meridian and how high, the clock time it crosses the elevation mask, which catalog anchors a flux estimate came from, and what went into its Min rec. If a number looks surprising, hover it before believing it.
- **Search**: type part of a name (`b0329`, `J0332`), or filter numerically — `dm<30`, `p<0.1` (seconds), `flux>10`, `alt>40`, or `magnetar`. Terms combine, so `dm<30 flux>50` finds bright, low-dispersion targets.
- **Include below mask**: also lists sources that are not up yet, and the **Next window** column says when each rises above the mask and how long the window lasts. (A source can be circumpolar — never setting — and still spend hours below a usable elevation.)
- **Include magnetars**: magnetars are marked ★ and are never removed by a flux filter, because the catalog usually carries no flux for them.
- **Flux at tuned freq**: estimates each source's flux *at the frequency you are tuned to* by power-law interpolation between the catalog's S400 and S1400 (using the source's own spectral index when both exist, a typical −1.6 otherwise) — labeled `est@…` so you know it is an estimate. Unchecked, the nearest catalog band is quoted verbatim.
- **Min rec**: the radiometer minimum recording length for an 8-σ folded detection at the current sample rate, from the site SEFD (`[site] sefd_jy`, measured on Cygnus A) and the catalog W50 pulse width (5% duty assumed when the catalog has none). **Rows highlighted green are viable now** — up, with Min rec fitting inside Time left. It is an aid, not a gate: one SEFD serves every band (low-band numbers read optimistic) and RFI, scintillation, and pointing loss add on top.
- **Best band**: of the bands the dish has feeds for (the Tuning presets), the one where this source detects fastest — flux scaled to each band, SEFD scaled by sky temperature, and pulse broadening from channel DM smearing plus empirical interstellar scattering. Steep-spectrum low-DM sources are sent low; high-DM sources are kept high, where scattering has not destroyed the pulse. Approximate physics — the band to *try first*, not a guarantee.
- **Best f**: the same model with the feed list taken away — the frequency anywhere from 100 MHz to 6 GHz where the source detects fastest, i.e. what a feed built for it would want to be. Hover the cell to see how far the dish's best real band falls short (often not far: for most bright pulsars the 408 MHz feed is within a few tens of percent of the optimum). An answer at the very bottom of the range means "as low as you can go", not a measured optimum — real pulsar spectra turn over there and the model does not know it.
- **Copy for reports**: Ctrl+C copies the selected rows (with a header line, and a comment line naming the site, the instant, the mask and the tuning the numbers came from) as tab-separated text that pastes cleanly into email, Excel, or Word; right-click offers Copy cell / Copy rows / Copy whole table.
- **What do I need?**: solves the dispersion arithmetic backwards for the selected source — which of the dish's bands (and how much bandwidth) would make its *DM measurable*, and what to set in the self-test

simulator. Dispersion delay goes as 1/frequency², so a small DM at L-band is simply unresolvable: the fold still detects the pulsar, but its DM search slides toward zero and means nothing. The answer says so plainly rather than leaving you to discover it after the drive.

- The catalog is downloaded once and cached beside your recordings, so the planner keeps working at a site with no internet. **Observe → Refresh Pulsar Catalog** re-downloads it.
- If a **Source** and a **Record for** duration are both set, the app warns at record time when that pulsar would set before the recording finishes.

Observe menu — B210 Self Test

A built-in test of the *entire* pulsar chain — SDR, channelizer, filterbank writer, timebase, PRESTO fold, verdict — with no test equipment at all. The B210's own transmitter plays a synthetic pulsar (100 ms period, DM 50, noise-carrier pulses with real cold-plasma dispersion) on 420 MHz, far from the protected hydrogen-line band; the receiver records the B210's internal TX→RX leakage — no cable or attenuator needed. The test first **measures that leakage path** with a tone and raises the TX gain in 10 dB steps (0–40 dB, microwatts into an open connector) only until the tone stands 30 dB above the receiver floor; the level it achieved is in the result. If even 40 dB falls short the result says so, and a loopback cable from TX/RX-A to RX2-A through a 30 dB pad gives a full-strength test. After the capture (default 90 s) the app folds the recording at the injected period and DM and grades PASS/FAIL: the period must come back exact, the DM near 50 (a DM stuck at 0 means dispersion was lost), and the significance high. Your tuning, sample rate, gain, and antenna are saved before the test and restored right after the capture, while the fold runs. Run it before packing for a field session: a PASS means a real pulsar that reaches the feed will survive the pipeline. Requires a USRP B200/B210 and an installed PRESTO. The recording and its fold PDF land in a `self_test` folder inside your recordings folder.

Beyond the standard test, two advanced modes make it a general pulsar *simulator*: **Simulate a catalog pulsar** picks any source from the ATNF catalog (magnetars included) and injects its exact catalog period and DM, and **Custom** opens every parameter — frequency (any B210 frequency, 70–6000 MHz), sample rate, period, DM, duty cycle, amplitude, RX gain, channels, and capture length. A live readout translates the chosen geometry into what matters: the dispersion sweep across the band and the DM resolution it can honestly support, pulse width vs sample time, and pulses per capture — narrow bands at high frequency constrain DM weakly, and the readout says so before you spend the time. After picking a catalog source, **Suggest geometry** goes further and solves for settings that can actually measure that source's DM, filling them in for you (it knows the internal leakage weakens at low frequency and that the duplex transmitter holds its timing best at or below 2 MS/s). TX gain is always locked at minimum: the internal leakage is all the test needs, so even the protected 1420 MHz band is safe.

Observe menu — Drift-Scan Review

The morning-after quicklook for drift-scan recordings: open one or more ezRA-format `.txt` files (written by the Workbench *or* by ezRA's own ezCol — both dialects are read, including ezCol's interleaved reference rows and mid-file elevation changes) and get one standard report per file. It checks the header (software version, whether the hydrogen line is inside the recorded band, the site coordinates against your settings, and whether the file's az/el disagrees with the Recording panel's current values — the classic stale-header trap), the recording itself (row cadence, gaps, total-power stability), and the spectrum (narrow spurs against a median bandpass, with the known 1420.000-family carriers called out). The plots show a bandpass-flattened waterfall, the averaged velocity profile, and the transit light curve with a Gaussian fit — peak, width, and the transit center in UT and local sidereal time, with the half-integration timestamp correction applied. Load several nights at once and the Trend tab plots the fitted peak day by day — the pointing/sag monitor used on the Haswell dish. **Copy summary** puts the whole text report on the clipboard; **Save report…** writes it to disk with PNGs of the plots. The review is deliberately the layer *below* ezRA's ezCon/ezSky/ezGal: it answers "did I detect it, and is the file clean?" so you know what is worth feeding onward.

Observation

The "what are you trying to do tonight?" selector. Pick a goal and every science-critical setting — band, sample rate, recording format, channels, integration — is set to a validated bundle in one step:

- **Pulsar — L-band**: 16 MHz, filterbank, 2044 channels, Integrate 1 (127.7 µs samples) at the L-band feed (1420.4 MHz) — the geometry behind the 28σ B0329+54 detection at Haswell.
- **Pulsar — UHF**: 20 MHz, filterbank, 256 channels, Integrate 16 (204.8 µs) centered at 420 MHz — the proven Haswell UHF geometry.
- **Magnetar / high-DM**: L-band with 4096 channels — narrower channels tolerate the larger dispersion of magnetars and distant pulsars.
- **Hydrogen line — drift scan**: ezRA .txt format at 1420.406 MHz, 2 MHz span, with the receiver's DC artefact moved out of band by an LO offset. On radios with no LO shift stage (HackRF, RTL-SDR) the offset cannot be applied in hardware, so the preset adapts: if the radio can run 8 MS/s it centers 1422.0 MHz instead — the line stays in band and the artefact sits 1.6 MHz away from any galactic gas (a configuration proven in the field by Ray Uberecken, AA0L); a radio that cannot reach that rate keeps the line-centered tuning with the artefact blanked in analysis. The status bar states which arrangement was applied. Set the dish Az/El in the Recording group. Starting an ezRA recording whose band does not contain the hydrogen line asks for confirmation first (and the status bar warns as soon as the tuning goes off the line).
- **RFI survey — sweep**: switches to Sweep mode; set the range in the Sweep group.

- **Manual (expert)**: touches nothing. The combo drops back here by itself when you change any of the settings a preset controls — the label never claims a bundle the settings no longer match. The app always starts here; your individual settings persist on their own.

Below the selector, a live **consequences line** translates the current settings into what they mean for the data: time resolution, channel width, per-channel dispersion smearing (at a reference DM of 30), and disk usage per hour. It turns amber when a combination is risky — sample rate beyond the validated recording geometry, or time resolution too coarse for pulsar work. On radios that can't reach a preset's rate, the request is clamped and snapped as usual and the readout shows what you actually got. Display settings (FFT size, window, averaging) are deliberately untouched by presets: they shape what you *see*, never what is recorded.

A preset never starts a recording. It only configures — review the settings, make any adjustments, then start the recording yourself with the **Record** control when you're ready. (The RFI-survey preset does begin sweeping the display immediately, exactly as the Sweep mode button would, but nothing is written to disk.)

Mode

- **Live**: real-time FFT of the radio's instantaneous bandwidth around one tuned center frequency (the traditional view).
- **Sweep**: stepped scan across a wide range — for RFI surveys that span more than the radio's instantaneous bandwidth. The radio is retuned across the range and each step's FFT is stitched into one wide trace. See the **Sweep** group below. Unavailable in Playback.

Tuning

- **Pulsar Band**: preset frequencies for common pulsar observation bands. Choose *Manual* to use the Manual Frequency field instead.
- **Coarse Tune**: ±100 MHz offset from the selected preset (or from the manual frequency).
- **Fine Tune**: ±10 MHz offset, layered on top of Coarse Tune.
- **Manual Frequency**: used when the *Manual* preset is selected. Accepts engineering notation, e.g. `1.42G` or `408M`.
- **LO Offset**: parks the receiver's local oscillator this far from the displayed center; the receiver's digital downconverter shifts the band back, so the display and every recorded frequency are unchanged — but the receiver's own DC spike (present in any zero-IF SDR, and sitting exactly on the displayed center by default) moves off your target. Set it beyond half the sample rate to push the spike out of the recorded band entirely. Vital for narrow-line work: tuned to the hydrogen line with offset 0, the spike lands *inside the line*. The *Hydrogen line — drift scan* observation preset sets +1.5 MHz automatically. Avoid offsets at integer multiples of the sample rate — on the B210 those raise a spur in the band (bench-measured).

Requires hardware support (B200/B210: yes; Soapy radios: only if the driver exposes a shift stage — HackRF and RTL-SDR do not, and the app falls back to classic tuning with a status-bar note).

The Tuning group is disabled in Sweep mode (the center frequency is chosen automatically per step).

Sweep (visible in Sweep mode)

- **Start / Stop**: bottom and top of the swept range. Keep them within the connected radio's tuning range.
- **Step**: Hz per tuning step. Type `auto` for ~80% of the current sample rate (recommended — keeps the clean middle of each FFT and avoids the DC spike plus band-edge rolloff). For tighter spacing enter a value, e.g. `2M`.
- **Status**: current step number and tune frequency, or *Idle* in Live.

Sweep runs continuously, redrawing the wide trace after each pass; the waterfall adds one row per pass. Sample rate and gain still apply to each step's FFT. Averaging, Max/Min hold, and baseline removal are forced off while sweeping (they would smear across retunes) and restored on return to Live; frozen cursor markers are cleared when entering or leaving Sweep.

RX

- **Sample Rate**: per-radio, up to the hardware's true maximum (e.g. B210: 0.625–61.44 MHz — 56 MHz analog bandwidth; SDRPlay: 2–10 MHz; RTL-SDR: 0.25–3.2 MHz). The presets are not arbitrary: each divides the radio's master clock evenly, so decimation stays on the flat half-band filter chain. Rates that don't divide the clock cleanly can fall back to CIC filtering, whose passband droop shows up as a bowl-shaped gain error across the spectrum — poison for calibrated radio astronomy. Higher rate = wider spectrum but more disk and host load when recording: the radio isn't the only limit. USB bandwidth and host CPU set a practical ceiling — if the overflow panel or a recording's gap counter climbs at a high rate, the host can't keep up; step down. (16 MHz .fil recording is the validated DSES pulsar geometry.) PlutoSDR-class radios (ADALM-Pluto, ANTSDR, the HamGeek E310) talk over USB 2.0 and report no overflows: above about 6 MS/s the stream silently loses samples, so the program offers them 6 MS/s at most, and a rate change on them briefly stops and restarts the stream (their driver rebuilds its buffer for the new rate).
- **Manual Rate (Hz)**: type any rate the SDR supports — useful for real-pulsar capture geometries that aren't in the preset list. The value is clamped to the device's reported min/max (hover for the range) and the radio snaps to the nearest rate it can actually deliver, which is then shown back. Accepts engineering notation (e.g. `24M`, `625k`). The combo clears when the active rate isn't one of the presets.
- **RX Gain**: per-radio range and meaning. The slider's min/max matches what the driver reports (e.g. B210: 0–76 dB on the AD9361 gain table; SDRPlay: 0–48 dB, internally inverted so higher = stronger

signal; RTL-SDR: 0–49.6 dB). AGC, if the driver defaults it on, is disabled at startup so the slider always takes effect.

- **Antenna**: appears only when the open radio has more than one RF input. For a B210 this lists all four physical connectors as `A : RX2`, `A : TX/RX`, `B : RX2`, `B : TX/RX` — receiver A and receiver B, each with its two SMA ports — and switching includes hopping between the two receivers. An RSPduo lists its two tuners. Pick the connector your cable is actually plugged into; the choice is remembered per radio. Single-port radios (most RTL dongles, the RSP1B) don't show this control.
- **Device**: shows the currently-open radio and re-opens the picker on click.

Recording

- **Folder**: where recordings land. Defaults to `~/Documents/DSES_SA_Recordings`. Change it with this button or from **File → Set Recording Folder…**; the folder is checked for writability and the next recording uses it immediately (it cannot be changed while a recording is running, so a run is never split across two folders). **File → Open Recordings Folder** opens the current one in your file manager.
- **Source**: optional source / pulsar name (e.g. `B0329+54`). When set it is folded into the recording filename and written into the SIGPROC `.fil` header (`source_name`, plus RA/Dec derived from the name) and the SigMF description, so PRESTO/prepfold pick it up. Blank gives a timestamp-only filename. Locked while recording.
- **Format**: *Raw I/Q (SigMF)* writes full-rate complex samples to a SigMF `.sigmf-meta`/`.sigmf-data` pair — exact, but large (e.g. ~192 MB/s at 24 Msps). *Filterbank (.fil)* channelizes the stream live and writes a SIGPROC filterbank (`telescope_id 12`) straight to disk, so the giant raw I/Q is never stored. The `.fil` is what PRESTO folds; it is produced by the same validated code as the offline `iq_to_fil.py` converter. *Drift scan (ezRA .txt)* writes integrated spectra (one row every ~10–15 s with the default geometry) in the ezRA data format, so the file feeds Ted Cline's free ezRA suite (ezCon → ezPlot/ezSky/ezGal) directly — hydrogen-line drift scans with the same radio that records pulsars. The filename follows the ezCol convention (`<prefix>YYMMDD_HH.txt`), and — also like ezCol — a multi-day recording rolls to a **new file at each UTC midnight** (one file per UTC day, each with a full header), so ezCon consumes a long campaign as its usual series of daily files.
- **Channels** / **Integrate** (filterbank only): the FFT channel count and how many power frames are summed per output sample, so `tsamp = channels × integrate / sample rate`. Locked while recording.
- **Az / El** (drift scan only): the dish pointing written into the ezRA file header (`azDeg`/`elDeg`). The observing-site identity (latitude, longitude, altitude, name) comes from the `[site]` section of the settings file — defaults are the DSES Haswell 60-ft dish. FFT bins, integration count, and the band-edge trim follow the dish's proven ezCol geometry and are adjustable via `[recording] ez_*` settings.

- **Record for**: optional fixed length — minutes (e.g. `30`) or `H:MM` / `HH:MM:SS` (e.g. `1:30`). The recording auto-stops when it is reached and the counter shows a countdown; blank records until you stop it. Locked while recording.
- **Start at**: optional scheduled start — a clock time (`03:15`, the next time it comes round), a delay (`+30` minutes, `+1:30`), or a full date (`2026-10-24 03:15`), read as UTC or your local clock per the selector beside it. Setting **Record** to *Recording* then *arms* instead of starting: the status turns amber, the counter shows a countdown to the start (and, with **Record for**, the planned end), the source-visibility and hydrogen-line questions are asked up front so the start itself needs nobody at the keyboard, and *Stopped* cancels. Blank starts at once.
- **Repeat**: make the same recording again and again with the start moving by itself. *Every sidereal day* spaces the starts 23 h 56 m 04 s apart, so each one begins 3 m 56 s earlier by the clock and the same patch of sky is overhead every time: the galactic plane crossing of a fixed dish, a pulsar, any source with a fixed place among the stars. *Every solar day* keeps the same clock time (the Sun). *Custom interval* takes your own spacing as `H:MM:SS` — `24:50:28` follows the Moon's average day, though the Moon and the planets wander against any fixed spacing over a few weeks. Each recording is **Record for** long (required here), **×N** sets how many, and the first start is **Start at** (or now when that is blank); every later start is counted from the first one, so the spacing never drifts. The amber status shows which recording is next (*3 of 14*) and when. If the computer wakes up inside a recording's window, the rest of that window is recorded and it still ends on schedule; a recording that cannot start (no radio) is skipped and the series goes on. *Stopped* cancels the whole series. The series lives in the running program: after a restart, put the first start's date and time back into **Start at** (hover the amber status to read it) with the same Repeat setting, and it resumes at the next one due.
- **Record**: *Stopped* / *Recording*. Recording always starts *Stopped* on launch. While recording, a red **REC** counter shows elapsed time (or the countdown when a duration is set).
- **Analyze when done** / **Quick look** (filterbank + PRESTO): when a `.fil` recording stops, the canned PRESTO pipeline runs automatically — `readfile` sanity, an `rfifind` RFI mask, band-edge zapping, and a `prepfold` fold — by catalog pulsar when Source is a known designation (the **Fold P (ms)** / **Fold DM** boxes then preview its catalogue values and lock, and the fold uses the pulsar's full ephemeris), or at a manual **Fold P (ms)** for a source with no catalogue entry (e.g. the lab pulsar simulator) — and delivers a **self-contained PDF** next to the recording: chart, commands, numbers, and a plain-language verdict. Verdicts are honest about failure modes: a periodicity that optimizes to DM ≈ 0 is reported as a *terrestrial signal*, not a detection, and "no detection" explicitly does not mean a bad recording. **Quick look** does the same on a snapshot of the still-growing file mid-recording, without interrupting it. Requires PRESTO (Mac/Linux: native install; Windows: WSL via `presto/build_presto.sh`).
- **Timebase integrity** (filterbank, USRP/UHD radios): if the host briefly can't drain samples (an RX overflow — the 'O' characters in the sidebar), the dropped stretch would silently shorten the file's sample

clock and smear a later pulsar fold. The recorder measures each gap from the radio's own timestamps and inserts zeros for exactly the missing time (to the nearest spectrum — the channelizer runs in the flowgraph since 1.7, so padding is in whole rows of one *tsamp*; every gap is measured against the recording's original time reference, so rounding never accumulates), and the `.fil` timebase keeps tracking real time. The REC counter shows any gaps live (e.g. *2 gaps, 45 ms padded*) and a `.gaps.json` file with the details is written next to the recording. Non-UHD radios (HackRF, RTL-SDR, SDRplay) don't provide per-gap timestamps; for them, watch the Overflow panel — a clean panel means a clean timebase.

Spectrum (top plot)

Live FFT magnitude in dB. The FFT itself runs inside the GNU Radio flowgraph (compiled blocks, since 1.7), so *every* sample the radio delivers is integrated into the trace at any sample rate, whether or not a recording is running — the display's statistics never change when you press Record, and the window stays responsive on small computers (the Raspberry Pi, an older laptop). Use the control panel to adjust:

- **FFT Size**: 256–65536. Larger = finer frequency resolution (RBW = sample rate / size) and a lower per-bin noise floor, but slower response. Changing it rebuilds the display chain — a brief pause, nothing else.
- **Window**: Blackman-Harris is the default — low sidelobes, good for RFI hunting. Hann/Hamming have narrower main lobes; Rectangular has the sharpest peak but the worst sidelobes.
- **Avg α**: exponential averaging across display updates (each update is already the mean of every FFT block since the previous one). 1.0 = no smoothing (every frame is a fresh measurement). Smaller = more smoothing; the ≈N readout is the effective number of frames averaged. For watching a weak line (the hydrogen line is a few tenths of a dB on a small dish) use ≈0.02–0.05 with **Baseline: Flatten** below.
- **DC suppress**: hides the zero-IF DC / LO-leakage spike that HackRF, RTL-SDR and similar radios put at the tuned centre by bridging the window's main lobe around the centre bin with its neighbours. Display only — the recorders always get the raw stream.
- **Trace**: how the trace is drawn when there are more FFT bins than screen pixels (any large FFT size). *Average* (the default) draws the mean power of the bins under each pixel: a thin line, the way SDR Console draws it, in which a weak broad line stands out without a long time average. *Peak* draws the highest and lowest bin under each pixel — a band as thick as the noise (at 65536 points it can be thicker than the hydrogen line is tall), but the weakest *narrow* carrier still shows, so use it for RFI hunting; Sweep always uses it, and so do the Max/Min hold traces. *Average + peak* draws the thin line over a faint peak band. Zoom in until there are two bins or fewer per pixel and every bin is drawn as it is, whatever this is set to.
- **Line view** (the button at the top of the panel, View menu, Ctrl+L): one click to *watch* a spectral line change in shape and height. It picks the FFT size that gives about 1 kHz resolution at the current sample rate, sets Avg α to 0.05 (about a two-second response), Smooth to 9 bins, Trace to Average, turns the

holds off, zooms to ±0.9 MHz around 1420.406 MHz when the hydrogen line is in a wide enough band, and a few seconds later fits the Y axis closely around the middle of the trace (a narrow carrier and the band edges may run off the plot rather than flatten the line). Click it again to get back the settings you had. Why these numbers: the scatter of a trace falls as 1/√(resolution × time), so a 30 Hz bin needs fifty times the averaging of a 1.5 kHz one for the same smoothness — fine resolution is for narrow carriers, not for a line a hundred kilohertz wide. Baseline is left as you set it; add **Flatten** or a stored reference if you want the bandpass removed as well.

- **Baseline**: removes the receiver's bandpass shape so a weak line stands up from a flat floor near 0 dB. *Off*: raw spectrum. *Reference (ON/OFF)*: point at empty sky, click **Store reference (OFF)**, then point at the source — the plot shows the difference. *Flatten (median)*: subtracts a wide running-median estimate of the bandpass in real time with no reference needed; set **Flatten width** several times wider than the feature you want to keep (a 1 MHz hydrogen profile at 8 MS/s and 4096 points is ~500 bins, so ~1501 bins). Either mode auto-fits the Y axis to the flattened trace. This plus a low Avg α is the "watch the line" view; the ezRA drift-scan recording is unaffected by any of it.
- **Max / Min hold**: overlay traces showing the highest/lowest value ever seen at each bin. Use **Reset** to clear. The **Detector** combo picks what they accumulate. *Peak (per FFT)*: the extreme of every FFT block — even a sub-millisecond burst registers at full amplitude, making Max hold a true transient-RFI catcher; on pure noise the Max trace settles ~10 dB above the average and the Min trace keeps sinking (the statistics of extremes over many samples — not a malfunction), so Min hold reveals what is *always* present: steady carriers stand up out of the collapsing noise floor. *Average (per frame)*: extremes of each display update's deep average — both holds stay within a few dB of the baseline, useful for tracking slow drifts.
- **Y-Axis**: dB min/max, or click **Autoscale** to fit the current data. **Reset Axes** snaps the plot back to the default dB range and full-span frequency view — handy after you've zoomed/panned with the mouse or nudged the min/max and want to get un-lost. It leaves FFT size, window, traces, and colors untouched.
- **Linear scale**: plots linear magnitude instead of dB (the default log scale). In linear mode the Y axis auto-fits and the dB Min/Max boxes are disabled. Affects the spectrum plot only — the waterfall stays in dB.
- **Trace**: color, line width, alpha, label.

Waterfall (bottom plot)

Scrolling 2-D image of FFT vs. time. Newest row at the **top**, history flowing down (the SDR#/GQRX convention); the left axis reads as age in rows.

- **Intensity Min/Max**: dB range that maps to the colormap. **Autoscale intensity** picks the 5%–99% percentile of the current data.
- **Colormap**: viridis (default), plasma, inferno, magma, turbo, cividis, gray.
- **Rows**: how many history rows to display (default 256).

Persistence

All selections are saved to a plain-text INI file and restored on next launch. The file location is shown in **Help → About**; you can open the folder directly with the **Open Settings Folder** button.

To revert everything to factory defaults, use **Restore Defaults** in the About dialog.

Update checks

If the developer has configured a manifest URL, the program checks for a newer release in the background at launch (no more than once every 24 hours). When a newer version is found, a non-modal dialog opens with the release notes and a button that opens the download page in your browser — you can ignore it and keep using the app, or click **Skip this version** to not be reminded about that particular version again.

The check is read-only and never auto-downloads or auto-installs. To trigger a check manually, use **Help → Check for Updates…**. To disable auto-checks, set `auto_check = false` under `[updates]` in the settings file. If the manifest URL has not been configured yet, the auto-check is silently skipped. Installs upgraded from 1.3.4 or earlier keep working: their stored manifest URL (the old `b210_sa` release folder) is moved to the new `dses-workbench` folder automatically on first launch, and the old folder keeps a pointer to the current release.

Tips for pulsar work

- The 1420.4 MHz preset tunes the L-band feed directly to the neutral-hydrogen line (1420.406 MHz), so HI stays in band at any sample rate. For drift scans prefer the **Hydrogen line** Observation preset, which also sets the rate, format and LO offset.
- 1666 MHz preset covers the OH maser band.
- Use **Avg α** ≈ 0.05 and **Max hold** to find intermittent RFI sources.
- The waterfall reveals time-structured interference (e.g. radar sweeps, ADS-B bursts) that the live spectrum smears out.