

SMACC

User manual

Remington Mallett

Contents

1	SMACC	9
1.1	What SMACC does	9
1.2	Get started	9
1.3	Used by	9

I Getting started

2	Installation	13
2.1	Updating SMACC	13
2.2	Development build (experimental)	14
2.3	After installing	14
3	SMACC files	15
3.1	Creating a SMACC file	15
3.2	The data directory	16
3.3	Opening a SMACC file	16

II Running a session

4	Overview	19
4.1	Opening SMACC	20
4.2	The Session window	21
4.3	The tools	22
4.4	Event log	22
4.5	Display preferences	22
4.6	SMACC files	23
5	Audio cues	25
5.1	The cue board	25
5.2	Designing a cue	25
5.3	Background noise	26
5.4	Is the cue reaching the bedroom?	26
6	Visual cues	29
6.1	The visual cue board	29
6.2	Patterns	29
6.3	What the markers mean	30
6.4	BlinkStick or Philips Hue?	30

6.5	Safety	31
6.6	Recipes	31
6.7	Known limitations	31
6.8	Troubleshooting	31
7	Biocals	33
7.1	Running a biocal	33
7.2	Sequences	34
7.3	Markers	34
7.4	Voice recordings	34
8	Dream reports & surveys	35
8.1	Recording a dream report	35
8.2	Surveys	35
8.3	Built-in surveys	35
8.4	Using a survey	36
8.5	Response files	36
8.6	Building your own	37
8.6.1	Item types	37
9	Intercom & chat	39
9.1	Voice	39
9.2	Text chat	39
10	Markers & port codes	41
10.1	What are port codes and triggers?	41
10.2	Terminology	41
10.3	Default event codes	41
10.4	Configuring codes	44
10.5	Event logging panel	44
10.6	Where codes live	44
10.7	How SMACC sends triggers	45
10.7.1	What is the baud rate?	45
10.8	Pulsed vs. set-and-hold	45
10.9	Configuring trigger output in SMACC	46
10.10	Finding your parallel-port address	46
10.11	Installing the InpOut32 driver (parallel port only)	46
10.12	Verifying	47

III

Devices & routing

11	Audio routing	51
11.1	Equipment and actions	51
11.1.1	No “system default”	52

11.1.2	Monitoring routes	53
11.2	One audio engine	53
11.3	Monitoring and volume	53
11.4	Windows only	54
12	Compatible devices	55
12.1	BlinkStick	55
12.1.1	What you need	55
12.1.2	Using it in SMACC	55
12.2	Philips Hue	55
13	Volume & latency	57
13.1	Volume	57
13.2	Latency	57
13.3	What contributes	58
13.4	The marker marks the software event, not the photons	58
13.5	Measured: a typical Windows rig	59
13.6	The Low / High setting	59
13.7	Visual cues: BlinkStick vs Hue	59
13.8	Measure it on your own rig	59
13.9	If you want it lower	60
14	EEG Annotator	61
14.1	Supported recordings	61
14.2	Viewing	61
14.2.1	Keyboard navigation	62
14.2.2	The epoch model	62
14.3	Annotating	62
14.4	Sleep staging	63
14.5	Session log overlay	63
14.6	Multiple raters	64
14.7	Blind-rater mode	64
14.8	For developers	65
15	Troubleshooting	67
15.1	Common problems	67
15.2	If SMACC crashes	67
15.3	Reporting a problem	67
16	Release notes	69
16.1	0.1.1 — 2026-06-15	69
16.2	0.1.0 — 2026-06-15	69

17	File formats	73
17.1	Where each file lives	73
17.2	Stability promise	73
18	SMACC file (.smacc)	75
18.1	On-disk shape	75
18.2	Envelope	76
18.3	settings	76
18.3.1	Audio, noise, visual, survey, chat, volume	76
18.3.2	biocals	77
18.3.3	Window-level	78
18.3.4	event_codes	78
18.3.5	devices	78
18.3.6	trigger_output	78
18.3.7	hue	79
18.4	Paths and portability	79
18.5	Version history	79
19	Preferences file (preferences.yaml)	81
19.1	Example	81
19.2	Fields	81
19.2.1	preferences	81
19.2.2	EEG Annotator keys	82
19.3	Version history	82
20	Session log (.log)	83
20.1	Line format	83
20.2	Log levels	83
20.2.1	Stimulus marker timing	84
20.2.2	Text-chat transcript	84
20.3	Embedded settings blocks	84
21	BIDS export (events.tsv + sidecar)	85
21.1	events.tsv	85
21.2	JSON sidecar	85
22	Annotations sidecar	87
22.1	The TSV	87
22.2	The JSON	87
22.3	Multiple raters	88
22.4	Precedence	88
23	Contributing	89
23.1	Conventions	89
23.2	Commit and pull-request style	89
23.3	Development	90
23.3.1	Tests	90

23.4	Building the executable	90
23.5	Building the docs	91
23.6	Versioning	91
23.7	Project notes	92

1. SMACC

Sleep Manipulation And Communication Clickything (SMACC) is a Windows desktop app for running sleep and dream studies — presenting cues to a sleeping participant, communicating with them, marking events on the EEG, and collecting dream reports.

In a dream-engineering session the experimenter has to deliver a precisely-timed cue to a sleeping participant, mark each event so it lines up with the EEG, talk with the sleeper, and collect a dream report — all from a dark control room, late at night, on whatever hardware the lab has. SMACC is the clickable control surface for that work: one window per job, glanceable in the dark, with a volume safety cap and explicit device routing so a misclick can't blast or mislead a sleeping participant.

[Download SMACC Installation guide](#)

The button downloads the installer (`SMACC-Setup.exe`) for the latest version, which installs per-user (no admin rights) and runs on 64-bit Windows 10 or later. See [Installation](#), which also covers the Windows SmartScreen warning on the unsigned download and how to get older versions.

Working offline? [Download these docs as a single PDF](#).

1.1 What SMACC does

- Trigger **audio cues** (and design them in-app with the Audio Cue Designer)
- Trigger **visual cues** on a BlinkStick or Philips Hue light
- Play masking **background noise**
- Run **biocalibrations** as timed, marked tasks
- Record **dream reports** and administer **surveys**
- **Talk, listen, and type** with the participant over an intercom
- Mark events with **EEG port codes** over LSL or a hardware TTL trigger
- Review and score recordings in the **EEG Annotator**
- Save a detailed **event log** for every run

1.2 Get started

Getting started

- [Installation](#) — download and run SMACC.
- [SMACC files](#) — create and reuse a study's configuration (`.smacc`).

Running a session

- [Overview](#) — the Launcher, the Session window, and the tools.
- [Audio cues](#) · [Visual cues](#) · [Biocals](#) · [Dream reports & surveys](#) · [Intercom & chat](#) · [Markers & port codes](#)

Devices, volume & timing

- [Audio routing](#) · [Compatible devices](#) · [Volume & latency](#)

After the night

- [EEG Annotator](#) — review and score recorded EEG.
- [Troubleshooting](#) · [Reference](#) · [Contributing](#)

1.3 Used by

SMACC is used in dream engineering research, including by:

- [Ken Paller's Cognitive Neuroscience Lab](#) at Northwestern University
 - Torres-Platas et al. (2026). Intentional lucid dreaming with a transformative learning agenda. *Research Square* doi:[10.21203/rs.3.rs-8745420/v1](https://doi.org/10.21203/rs.3.rs-8745420/v1)

- Konkoly et al. (2026). Using real-time reporting to investigate visual experiences in dreams. *J Cogn Neurosci* doi:[10.1162/jocn.a.107](https://doi.org/10.1162/jocn.a.107)
- Morris et al. (2026). Inducing lucid dreaming based on a contemplative practice of compassion. *Brain Sci* doi:[10.3390/brainsci16030315](https://doi.org/10.3390/brainsci16030315)
- Konkoly et al. (2026). Creative problem-solving after experimentally provoking dreams of unsolved puzzles during REM sleep. *Neurosci Conscious* doi:[10.1093/nc/niaf067](https://doi.org/10.1093/nc/niaf067)
- Konkoly et al. (2025). Investigating dreams by strategically presenting sounds during REM sleep to reactivate waking experiences. *Neuropsychologia* doi:[10.1016/j.neuropsychologia.2025.109229](https://doi.org/10.1016/j.neuropsychologia.2025.109229)
- Morris et al. (2025). Lucid dreaming of a prior virtual-reality experience with ego-transcendent qualities: A proof-of-concept study. *Neurosci Conscious* doi:[10.1093/nc/niaf017](https://doi.org/10.1093/nc/niaf017)
- Mundt et al. (2024). Treating narcolepsy-related nightmares with cognitive behavioural therapy and targeted lucidity reactivation: A pilot study. *J Sleep Res* doi:[10.1111/jsr.14384](https://doi.org/10.1111/jsr.14384)
- Wolk et al. (2024). Lucid dreams from reactivating mindfulness during REM sleep: A pilot study. *Int J Dream Res* doi:[10.11588/ijodr.2024.2.98233](https://doi.org/10.11588/ijodr.2024.2.98233)
- [Michelle Carr's Dream Engineering Lab](#) at the University of Montreal and the Center for Advanced Research in Sleep Medicine
 - Jafarzadeh Esfahani et al. (2024). Highly effective verified lucid dream induction using combined cognitive-sensory training and wearable EEG: A multi-centre study. *bioRxiv* doi:[10.1101/2024.06.21.600133](https://doi.org/10.1101/2024.06.21.600133)

SMACC is free software released under the [GPL-3.0-or-later](#) license.



Getting started

2	Installation	13
2.1	Updating SMACC	13
2.2	Development build (experimental)	14
2.3	After installing	14
3	SMACC files	15
3.1	Creating a SMACC file	15
3.2	The data directory	16
3.3	Opening a SMACC file	16

2. Installation

Download SMACC

The button above always downloads the installer for the **latest** version — double-click `SMACC-Setup.exe` and click through. The installer:

- installs SMACC **per-user** — no administrator rights or IT involvement needed;
- adds a **Start menu** entry (and, if you opt in, a desktop shortcut);
- associates **.smacc files**, so double-clicking one opens the **Start a session** dialog with it preselected;
- includes the **EEG Annotator** — the post-hoc [EEG viewer/annotator](#), opened from the Launcher;
- registers an uninstaller — remove SMACC from **Settings > Apps** like any other program. Uninstalling never touches your data: SMACC files, recordings, and logs under `~/SMACC` (or your data directories) all stay put. The uninstaller reminds you of this (and of the folder's location) when it finishes, so finding the folder afterwards is expected — delete it manually if you no longer need the data.

Installing a newer version over an existing one upgrades it in place.

To get an older or specific version, browse the [releases page](#): pick the release you want, open its *Assets* dropdown, and download from there.

i System requirements

SMACC runs on 64-bit Windows 10 or later.

⚠ Windows SmartScreen — “Windows protected your PC”

SMACC isn't code-signed yet, so when you run the installer Windows SmartScreen may show a blue **“Windows protected your PC”** box. This is expected for any new, unsigned program. To proceed anyway, click **More info**, then **Run anyway**.

i Administrator privileges and the UAC prompt

For some features you will need to open SMACC with Administrator privileges (right-click the Start menu entry and select **Run as administrator**). Windows then shows a **User Account Control (UAC)** prompt asking whether to allow the app to make changes — click **Yes** to continue. For everyday use (audio cues, dream reports, LSL markers) you can run SMACC normally, without administrator rights.

i For IT departments

The default install is per-user (under `%LOCALAPPDATA%\Programs\SMACC`, no elevation). On managed machines where per-user installs are blocked — e.g. AppLocker policies on `%LOCALAPPDATA%` — run the same installer machine-wide instead: `SMACC-Setup.exe /ALLUSERS` (elevates, installs to Program Files).

2.1 Updating SMACC

From the Launcher, **File > Check for updates...** asks GitHub whether a newer release exists and, if one does, offers to open the download page in your browser — run the new installer and it upgrades the existing install in place. SMACC never checks on its own: lab machines are often offline, and studies usually pin one version for their whole run, so checking is always an explicit click.

2.2 Development build (experimental)

Every code change merged to `main` is rebuilt and published as a single rolling **development build**, so you can try an unreleased fix or feature before the next stable release.

[Download the development build](#)

Unzip it and run `SMACC.exe` from the extracted `SMACC` folder. It's portable, so it runs without installing and leaves any installed copy of `SMACC` untouched.

⚠ For testing only

The development build is unsigned, unreleased, and **not tested**, and it is overwritten every time `main` changes — so it's a moving target, not a fixed version. Don't use it to run a study; pin a stable release for that. The **Download SMACC** button at the top of this page and the in-app **File > Check for updates...** always point at the latest *stable* release, never this build.

2.3 After installing

By default `SMACC` stores everything under `~/SMACC`. To use a different location, set the environment variable `SMACC_DIRECTORY` to the directory you want; `SMACC` creates it and its subfolders on first run.

The rest of setup is covered on the relevant pages:

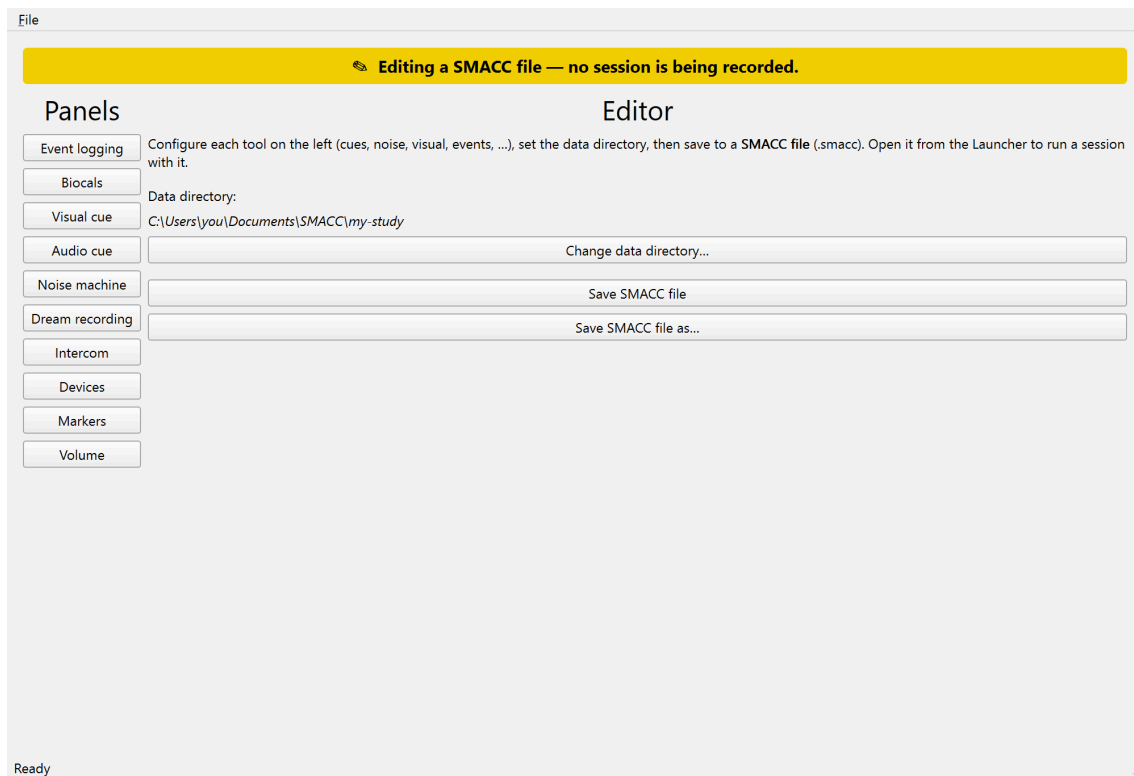
- **Your study configuration** lives in a portable [SMACC file](#). `SMACC` seeds a `default.smacc` and opens it when you don't pick another, so it works out of the box. The installer associates `.smacc` files; enable or repair the association any time from **File > Associate .smacc files (Windows)** in the Launcher.
- **Audio cues** go in the data directory's `cues/` folder, alongside the seeded `demo-*` cues — see [Audio cues](#).
- **Dream-report surveys** are managed from the Dream recording panel — see [Dream reports & surveys](#).
- **A recording mic**: to record dreams, bind your mic to **Bedroom mic 1** in the **Devices** window (the **Panels** column) — see [Audio routing](#).

3. SMACC files

A **SMACC file** (`.smacc`) is the study's configuration: cue files, volumes, noise, visual cues, biocals, survey presets, event markers, the display choices that apply to a session (always-on-top and log-preview levels), and the **data directory** where runs are written. It plays the same role for a SMACC study that a montage file plays for an EEG setup — configure it once, then reuse it session after session.

A SMACC file is *not* a session file. It is not tied to any particular run: opening one **starts a new session** with that configuration, and you can start as many sessions from the same file as you like. Sessions never rewrite the SMACC file they were started from — what happened in a run is recorded in that run's [session log](#), not in the SMACC file.

It is a single, portable file (plain YAML you can read and edit). Cue/noise files and the data directory are stored **relative** to the SMACC file when they sit beside it — so a self-contained study folder stays valid if you move, copy, or zip it — and **absolute** when they point elsewhere. For the exact on-disk format, see the [SMACC file reference](#).



3.1 Creating a SMACC file

There are two ways to make one:

- **From scratch, in the Editor.** In the [Launcher](#), click **Editor...** In the dialog, choose **New SMACC file** for a blank configuration (or pick an existing file to edit). Configure each panel on the left, set the data directory, and **Save SMACC file as...** to a new `.smacc` anywhere you like. The Editor never records a run — it only authors configuration.
- **From a live session.** In a running **Session**, use **File > Save SMACC file as...** to snapshot the session's *current* settings — including any mid-session tweaks (volumes, device routing, event-code edits) — to a new SMACC file. Handy when you've tuned things live and want tonight's setup to be tomorrow's starting point.

SMACC ships a `default.smacc` in your SMACC directory as a working example. It is treated as a read-only template — saving over it redirects to Save-As — so there is always a known-good configuration to fall back on.

You can keep SMACC files anywhere: for instance one per participant (`peter.smacc`, `paul.smacc`, ...) or one per protocol, each pointing at whatever data directory you like (they can share one).

3.2 The data directory

Every SMACC file names a **data directory** — where its runs are written. Each session started from the file gets its own timestamped folder under that directory (e.g. `smacc-20260607-223015/`) holding the run's `.log`, dream-report recordings, survey responses, and any exports. Set it in the Editor (**Change data directory...**); choose it deliberately, since it determines where a night's data ends up.

3.3 Opening a SMACC file

Opening a SMACC file starts a new session configured by it:

- **Double-click the file** (Windows build): SMACC opens the **Start a session** dialog with that file preselected, so you confirm the subject/session and click **OK** to start. Enable or repair the file association any time from **File > Associate .smacc files (Windows)** in the Launcher.
- **From the Launcher**: click **Session...**, then pick the file in the **Start a session** dialog (recent files are listed; **Browse...** finds any other) and click **OK**.
- **From a terminal**: `SMACC path/to/file.smacc`.

II

4	Overview	19
4.1	Opening SMACC	20
4.2	The Session window	21
4.3	The tools	22
4.4	Event log	22
4.5	Display preferences	22
4.6	SMACC files	23
5	Audio cues	25
5.1	The cue board	25
5.2	Designing a cue	25
5.3	Background noise	26
5.4	Is the cue reaching the bedroom?	26
6	Visual cues	29
6.1	The visual cue board	29
6.2	Patterns	29
6.3	What the markers mean	30
6.4	BlinkStick or Philips Hue?	30
6.5	Safety	31
6.6	Recipes	31
6.7	Known limitations	31
6.8	Troubleshooting	31
7	Biocals	33
7.1	Running a biocal	33
7.2	Sequences	34
7.3	Markers	34
7.4	Voice recordings	34
8	Dream reports & surveys	35
8.1	Recording a dream report	35
8.2	Surveys	35
8.3	Built-in surveys	35
8.4	Using a survey	36
8.5	Response files	36
8.6	Building your own	37
9	Intercom & chat	39
9.1	Voice	39
9.2	Text chat	39
10	Markers & port codes	41
10.1	What are port codes and triggers?	41
10.2	Terminology	41
10.3	Default event codes	41
10.4	Configuring codes	44
10.5	Event logging panel	44
10.6	Where codes live	44
10.7	How SMACC sends triggers	45
10.8	Pulsed vs. set-and-hold	45
10.9	Configuring trigger output in SMACC	46
10.10	Finding your parallel-port address	46
10.11	Installing the InpOut32 driver (parallel port only)	46
10.12	Verifying	47

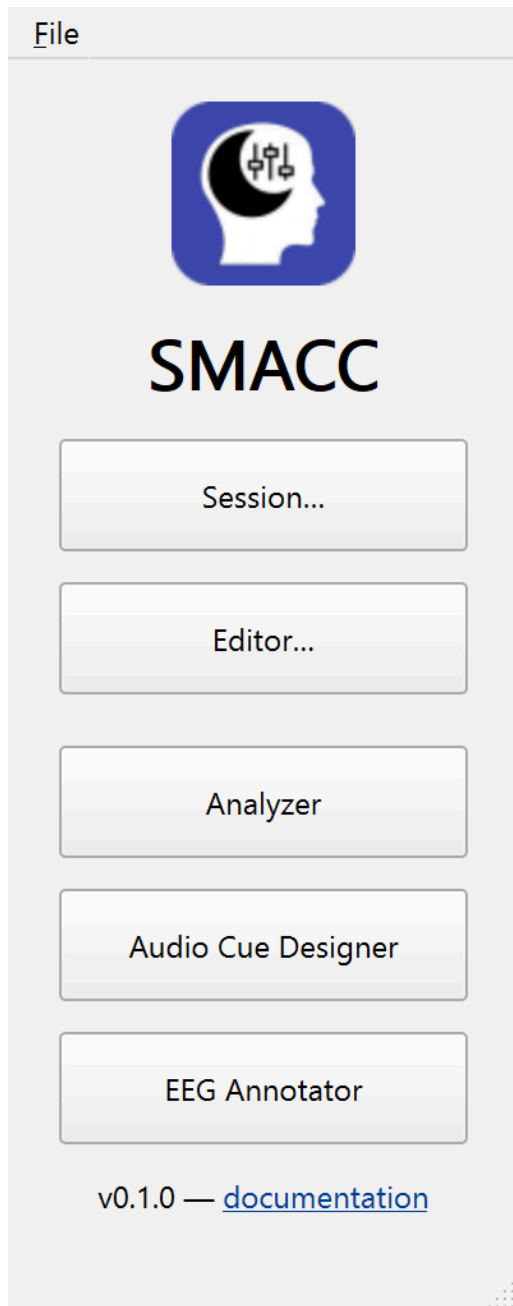
4. Overview

SMACC opens to its **Launcher** — a list of the SMACC tools. Click a tool and SMACC asks for the **SMACC file** to use, then opens it. To run a night you open a live **Session**. This page orients you; each tool then has its own page.

The Launcher's tools (its title bar reads **SMACC**):

- **Session...** — run a live **Session** for collecting data, the interface for running a night. SMACC prompts for a [SMACC file](#) first.
- **Editor...** — create or edit a [SMACC file](#) in the **Editor**, without recording anything.
- **Analyzer** — inspect a past session.
- **Audio Cue Designer** — build a tone cue and export it as a WAV.
- **EEG Annotator** — review and annotate a recorded EEG (see [EEG Annotator](#)).

The **Session** and the **Editor** are the same window in two modes — running a night versus authoring a SMACC file.



4.1 Opening SMACC

Open the app to get the Launcher. SMACC never drops straight into a session; the one exception is double-clicking a `.smacc` file, which opens the **Start a session** dialog with that file preselected. From the Launcher:

- **Session...** — opens the **Start a session** dialog. Pick the **SMACC file** (the picker lists recent files, the seeded `default`, and **Browse...** for any other), confirm the optional subject/session/notes, and click **OK** to start. Runs are written to that file's **data directory**; the run folder and log are created only when the session starts. (See [SMACC files](#) for what a SMACC file holds.)
- **Editor...** — opens the **Open in the Editor** dialog. Choose **New SMACC file** for a blank configuration, or an existing file (recents / **Browse...**) to edit. The Editor configures the panels (cues, noise, visual, markers, surveys), sets a data directory, and saves with **Save SMACC file as...**; it never records a run.
- **Analyzer** — open a past session (a `.log`, a session folder, or a zipped session) to see a summary (events, duration, subject/session, dream reports), export its events to a BIDS `events.tsv`, or recover its settings to a `.smacc` — all without starting a new session.

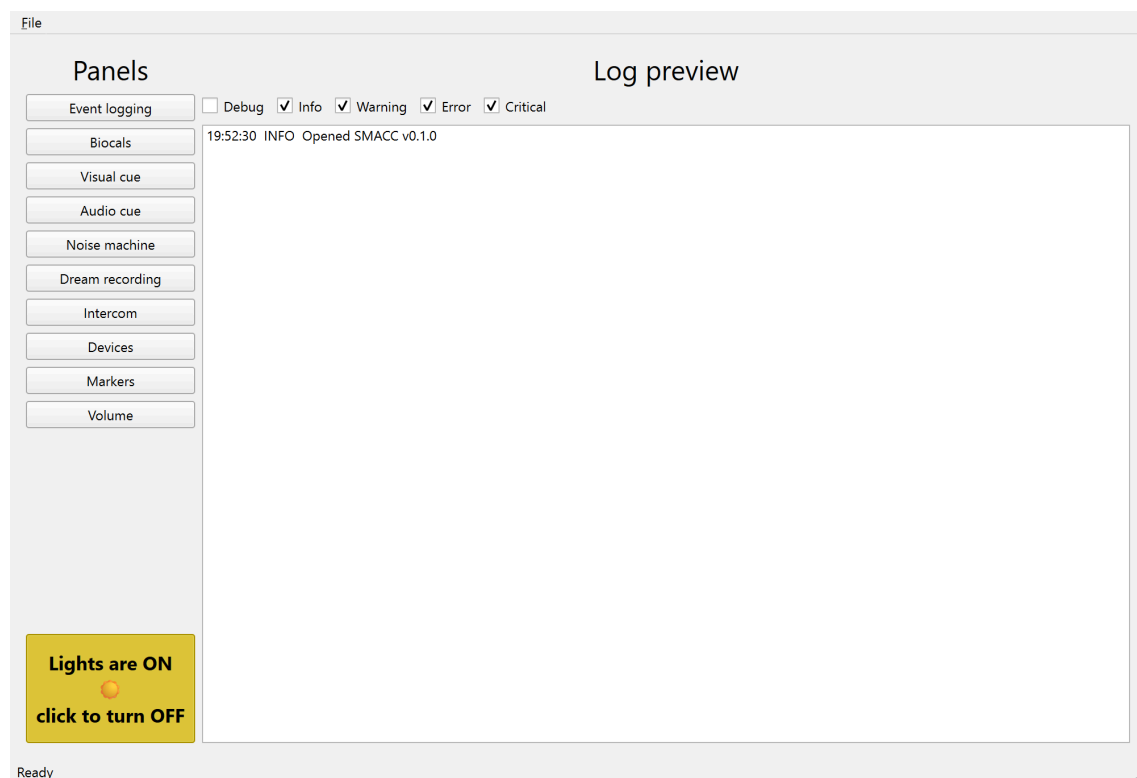
- **Audio Cue Designer** — open the standalone tool to build a simple tone cue and export it as a WAV into a study's [cues/](#) folder (see [Audio cues](#)).
- **EEG Annotator** — open the post-hoc [EEG Annotator](#) to review a recorded night and place annotations. It runs as its own window and process, so it stays available while a session runs.

Closing the Editor or a standalone tool returns you to the Launcher. Ending a Session quits SMACC entirely — the night is over. Closing the Launcher also quits SMACC.

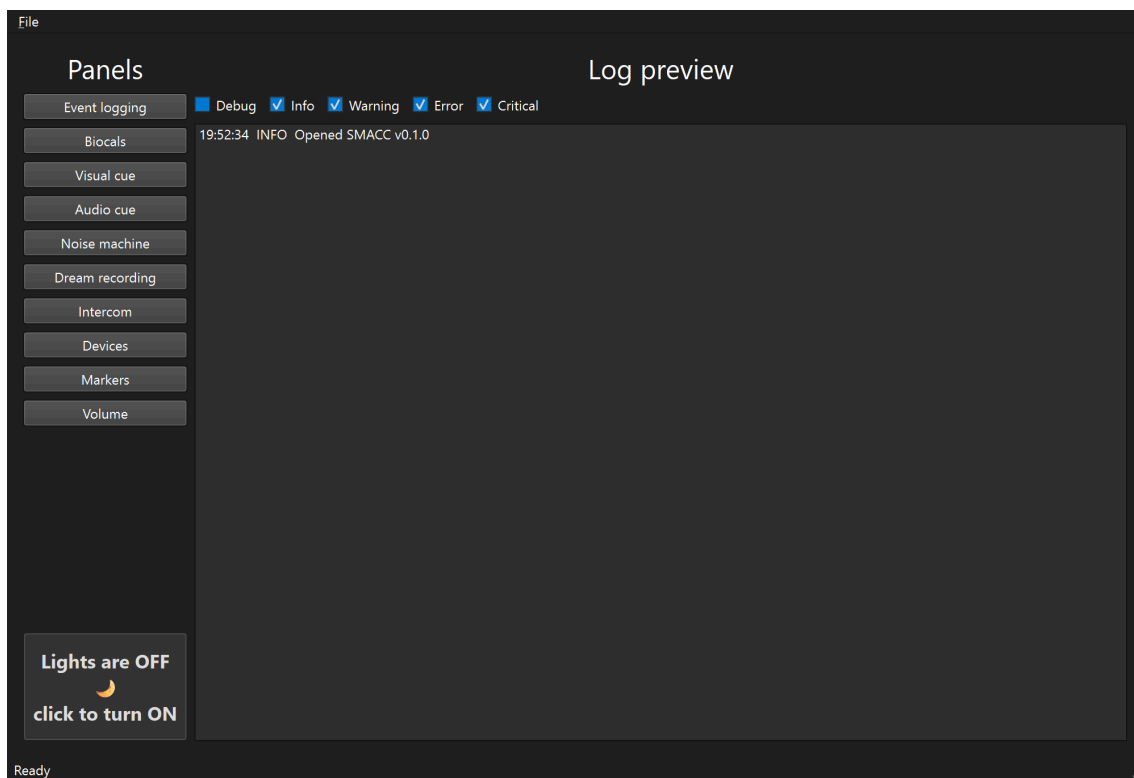
4.2 The Session window

A running Session has two columns. The **Panels** column on the left opens each tool window (Audio cue, Visual cue, Noise machine, Dream recording, Intercom, Biocals, Markers, Event logging, Devices, Volume). The right side is the live **Log preview**, with checkboxes to choose which [log levels](#) it shows. The **Lights** toggle on the main window dims the room (and flips the app's dark theme).

Tool windows close with **Ctrl+W** (or **File > Close window**), which only hides the window with its state intact; the session keeps running and the Panels column reopens it.



Flip **Lights** off and the whole app switches to a dark theme for the night — the same window, dimmed for a dark control room:



4.3 The tools

Each tool has its own page:

- [Audio cues](#) — the cue board, the Audio Cue Designer, background noise, and confirming a cue reaches the bedroom.
- [Visual cues](#) — light cues on a BlinkStick or Philips Hue.
- [Biocals](#) — timed, marked biocalibration tasks.
- [Dream reports & surveys](#) — record a report and administer a survey.
- [Intercom & chat](#) — talk, listen, and a typed channel.
- [Markers & port codes](#) — the Markers window, the Event logging panel, and EEG triggers.
- [Audio routing](#) and [Devices](#) — bind equipment and route actions to it.
- [Volume & latency](#) — the output safety cap and stimulus timing.

4.4 Event log

Every run writes a detailed `.log` to its own timestamped folder under the SMACC file's **data directory** (for example `~/SMACC/data/`), capturing the events and settings for that session. Open one later from the Launcher's **Analyzer** to see a summary, export its events to BIDS, or recover its settings. For the line format and levels, see the [session log reference](#). If SMACC crashes, see [Troubleshooting](#).

4.5 Display preferences

Some display choices apply to a session and travel with the study in the SMACC file: **always-on-top** (toggled per window — the Session window's **File** menu, each tool window's **File** menu, or **Ctrl+T** in whichever window is active) and which **log levels** show in the preview (the checkboxes above the log preview). Save them with the rest of the configuration from the Editor, or with **File > Save SMACC file as...** in a Session.

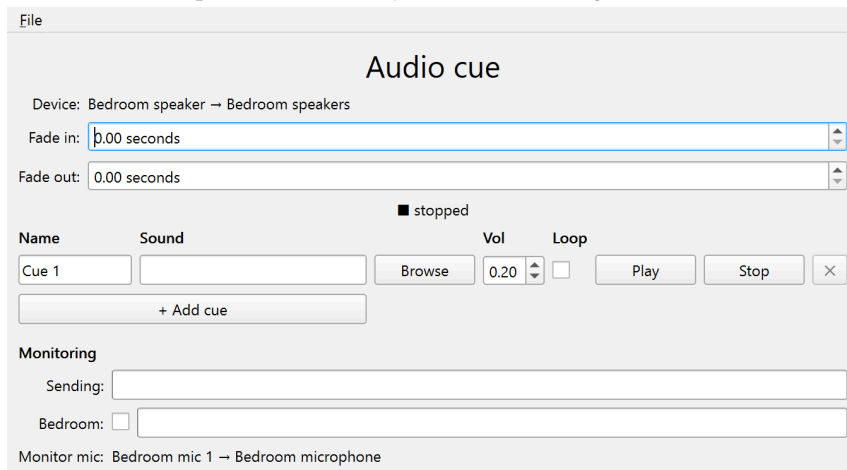
Separately, the machine remembers window positions and sizes and your recent files in `~/SMACC/preferences.yaml`, restored on the next launch (see the [preferences reference](#)).

4.6 SMACC files

A **SMACC file** captures your study's whole configuration — cue files, volumes, noise, visual cues, survey presets, event markers, display choices, and the **data directory** where runs are written — in a single portable `.smacc`. See [SMACC files](#) for the full story, and the [SMACC file reference](#) for the on-disk format.

5. Audio cues

Audio is SMACC's primary cueing modality. The **Audio cue** window (in the **Panels** column) holds the cue board: one row per cue, each ready to fire with a single click.



5.1 The cue board

Place sound files where your settings expect them — by default the data directory's `cues/` folder (for example `~/SMACC/data/cues/`; common formats such as `.wav`, `.mp3`, `.flac`, `.ogg`, and `.aiff` are accepted) — and trigger them from the cue controls. SMACC seeds a few `demo-*` cues there so there is always something to test with. You start with one cue, prefilled with a random demo, and use **+ Add cue** and each row's **×** to match a protocol (one minimum, up to 20).

Each play and stop is marked in the EEG record. The cue marker is stamped at the *estimated sound onset*, not the button press, so it lines up with what the participant hears (see [Volume & latency](#)).

5.2 Designing a cue

No sound file ready? Open the **Audio Cue Designer** from the Launcher to build a simple cue inside SMACC, with no external audio editor. Lay out a sequence of **tone** and **silence** segments (each tone has a frequency, duration, and level, with an optional bell-like **decay**), or start from a **preset** (a single chime, a pip train). The whole pattern can **repeat** ($\times N$ with a gap between repeats, the classic pip-train shape), and an optional whole-cue fade in/out and normalize finish it. A live **waveform** shows the cue as you edit, **Preview** plays it on your default output, and **Export WAV...** writes it into your study's `cues/` folder, where it appears in the cue board like any other cue.

The WAV is what the cue board plays; the *design* stays editable through **Save design...** / **Open design...** (a small `.json` file, kept alongside the WAVs by default), so you can reopen it tomorrow and nudge a level instead of rebuilding the cue. The designer is a standalone tool: it plays on the default device and ignores the session's device routing and volume safety cap.

Audio Cue Designer

Build a short cue from tone and silence segments, preview it, then export a WAV into your study's cues folder. Save the design to keep it editable.

Preset: Choose a preset...

Name: cue

Fade in: 0.00 s

Fade out: 0.00 s

☐ Normalize

Repeat: 1 x

Repeat gap: 0.00 s

Type	Freq	Duration	Level	Decay
Tone	440 Hz	1.00 s	0.50	☐

+ Add segment

Total length: 1.00 s



Preview Stop

■ stopped

Open design... Save design... Export WAV...

5.3 Background noise

The **Noise machine** window plays continuous masking noise on the cue route. Pick a **color** (white and the other generated colours) or point it at your own WAV (the **From file** option), set its **volume**, and start it. **Noise started** (62) and **Noise stopped** (63) mark the EEG. Noise shares the bedroom-speaker route and the output safety cap with cues, so calibrate it the same way.

5.4 Is the cue reaching the bedroom?

A cue you can hear in the control room is not proof the *participant* heard it: the bedroom speaker could be muted, unplugged, or turned down at the hardware. The Audio cue window's **Monitoring** section shows two meters so you can tell:

- **Sending** is the level SMACC is emitting to the cue output, the moment it plays. It is exact, but it only confirms SMACC is *playing*; it is blind to everything downstream (Windows volume, the speaker's power switch, the cable). Read it as a diagnostic: if *Sending* is dark, the problem is on SMACC's side (wrong cue, or a per-cue volume or the safety cap at zero); if it is lit but the room is silent, the problem is the speaker.
- **Bedroom** is the level a microphone actually picks up in the room. This is the objective check: it moves only when sound really happens in the bedroom. Tick the box beside it to start monitoring. Because a faint cue can sit close to a cheap mic's noise floor, the meter also shows the **rise above the room's resting level** (the **+N** next to the reading), so even a small bump stands out.

 A dedicated monitoring mic

The *Bedroom* meter listens on the **Monitor bedroom noise** route, which defaults to **Bedroom mic 1**. For the most reliable check — especially for the very quiet cues a study often starts at — bind a separate, sensitive **Bedroom mic 2** in the Devices window and route *Monitor bedroom noise* to it. That keeps verification independent of the (often cheaper, voice-activated) dream-report mic.

 A quiet mic isn't proof of silence

A cheap or voice-activated mic may not register a very faint cue even when it is playing, so a dark *Bedroom* meter is a prompt to check, not proof the cue failed. Read it together with *Sending*.

For how cue levels combine and how to calibrate them, see [Volume & latency](#). For binding the speaker and mics, see [Audio routing](#).

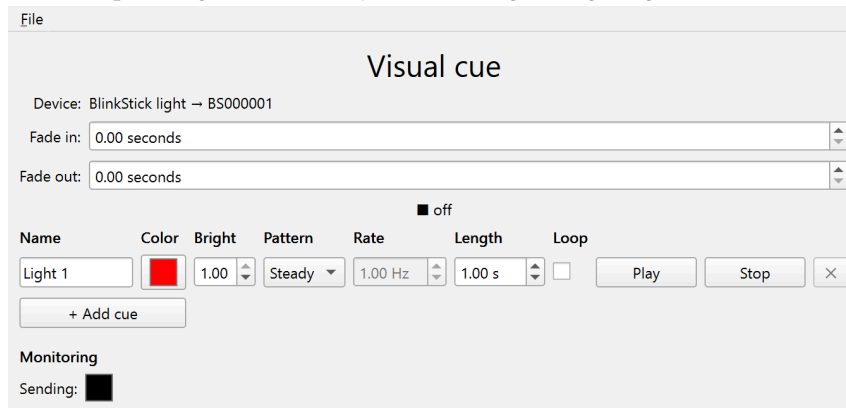
6. Visual cues

Light is SMACC's second cueing modality, and it earns its place for two reasons. Closed eyelids still transmit light, so a lamp can reach a sleeping (or dreaming) participant directly; flashing light during REM is the classic lucidity-induction signal. And light is silent, so it layers over masking noise or an auditory protocol without touching the soundscape. SMACC drives light cues on a USB **BlinkStick** or on **Philips Hue** bulbs, from one window, with the same marker discipline as audio.

This page covers the Visual cue board, the patterns, what the markers mean, choosing between the two devices, and the safety notes that come with flashing light at sleeping people. For plugging the devices in and binding them, see [Devices](#).

6.1 The visual cue board

The **Visual cue** window (in the **Panels** column) is the light sibling of the [Audio cue board](#): one row per cue, each kept configured and ready so that firing the right light at 3 a.m. is a single click.



Each row holds a **name** (it travels into the event log with every start/stop marker), a **color**, a **brightness** (0–1 — the visual volume), a **pattern** with its **rate**, a **length** in seconds, and a **loop** checkbox that keeps the cue going until you press its **Stop**. Use **+ Add cue** and each row's **X** to match the protocol (one cue minimum, up to 10).

Above the table, two shared fades shape every cue: **Fade in** ramps brightness up at the start, and **Fade out** ramps it down — both when you stop a cue early and when its length runs out, so a cue never cuts to black abruptly unless you want it to (leave the fades at 0 for crisp edges).

Playback is one light at a time: playing a row stops whatever else is lit (and marks its stop); re-playing the lit row just restarts it. While a cue is lit, its **brightness** and **loop** edits take effect immediately; color, pattern, rate, and length apply from the next Play.

The **Sending** swatch at the bottom shows the exact color SMACC is pushing to the device, frame by frame — the visual counterpart of the audio board's *Sending* meter. The same caveat applies: it confirms SMACC is *emitting*, not that the bedroom light actually lit. A BlinkStick has no return channel, so if you need proof, that's what the bedroom camera (or your own eyes) is for.

6.2 Patterns

- **Steady** — constant light for the length. With a long fade-in this is also the ambient/dawn workhorse (see [Recipes](#)).
- **Pulse** — a smooth brightness wave at the chosen rate (it starts dark and peaks mid-cycle, so onset is gentle). The usual choice for a salient-but-sleep-friendly cue.
- **Flash** — full on/off at the chosen rate (half on, half off). Maximal salience; the classic photic stimulus. BlinkStick only — see [the comparison](#) and [Safety](#).

The rate is set in **Hz**, the unit your methods section wants. The fades stay separate from the pattern: the pattern is the stimulus, and the fades only shape how it ramps in and out. Pattern timing is computed from elapsed time rather than accumulated per frame, so a busy GUI moment can’t drift the flicker.

6.3 What the markers mean

Every play and stop is marked in the log and on the trigger channel with the `VisualStarted` (66) and `VisualStopped` (68) events, each carrying the cue’s name. Two details matter for analysis:

- **VisualStarted fires after the first frame is committed to the device.** On a BlinkStick the USB write takes ~1–2 ms, so the marker trails the photons by about that much. On Hue, “committed” means the bridge accepted the command — the bulb itself transitions over the following ~100 ms, so the marker *leads* the photons by that lag (and it varies). Time-locked analyses should use the BlinkStick.
- **VisualStopped fires once the light is actually dark** — after the fade-out completes, not when Stop is pressed. The pair brackets the physical stimulus, not the button presses.

Every stop path turns the light off — Stop, the length running out, switching to another cue, a device failure, or quitting SMACC — so a cue can’t be left burning over a sleeping participant.

6.4 BlinkStick or Philips Hue?

	BlinkStick	Philips Hue
What it is	A USB stick/strip of RGB LEDs	Ordinary smart bulbs + a bridge
Connection	One USB cable, no network	Bridge on the rig’s LAN; bulbs via Zigbee
Buying	blinkstick.com only	Retail everywhere, but pricier
Setup	Plug in, bind, done	Pair once with the bridge’s link button (Devices > Philips Hue)
Cue onset	~1–2 ms after the marker	~100 ms of lag and jitter
Patterns	Steady, plus pulse and flash up to 20 Hz	Steady and slow pulse; flash is refused (the bridge rate-limits commands far below a square wave)
Coverage	A point source near the bed	Whole-room illumination, or a group of rooms
Fails when	The USB cable/port acts up	The bridge IP changes or the network drops

Rules of thumb: for **time-locked cueing** (EEG-aligned onsets, flicker at a defined Hz) use the **BlinkStick**. For **ambient light** — dawn simulation, whole-room color, slow breathing pulses — **Hue** is the better lamp. A rig can bind both and switch by re-routing *Play visual cue* in the Devices window.

i Hue on a university network

The bridge and the control PC must reach each other on the same network, and institutional Wi-Fi often blocks exactly that (client isolation, registration portals). The dependable setup is a small dedicated router (or an Ethernet drop) for the rig’s gear — and a DHCP reservation for the bridge, so its IP stops changing. The pairing key is stored in the study’s `.smacc` file; it only works from the local network, but treat the file accordingly.

6.5 Safety

⚠ Photosensitive epilepsy

Flicker between roughly 3 and 30 Hz can trigger seizures in photosensitive individuals, with peak sensitivity around 15–25 Hz. SMACC shows a warning on the board whenever any pulse/flash rate is set above **10 Hz** and caps the control at **20 Hz** — above that the USB timing couldn't deliver a faithful square wave anyway. Screen participants for photosensitivity and a seizure history before using flicker, and keep rates at or below 10 Hz unless the protocol specifically demands more.

For the sleeping side of the equation: the default cue color is **red** on purpose — melatonin suppression and arousal are driven mostly by short (blue) wavelengths, so red light is the gentlest way to be visible, which is also why it is the darkroom convention. Treat **brightness** like cue volume (start low and calibrate on a pilot night), and give cues a **fade-in** so they ramp rather than startle. An abrupt bright flash is how you buy an awakening and lose the data point.

6.6 Recipes

- **A TLR-style lucidity cue** — red, brightness ~0.2, **pulse** at 1 Hz, length 10 s, fade in/out ~1 s. Salient enough to penetrate REM, gentle enough to keep it.
- **Dawn simulation** — warm white, **steady**, length 300 s, fade in 60 s, on a Hue group: the room brightens like a sunrise for a gentler scheduled awakening.
- **Cue vs. sham** — two identical rows, the sham at brightness 0. The sham fires real `VisualStarted/VisualStopped` markers with zero light, so the trigger channel shows both conditions while only one stimulates.

6.7 Known limitations

- A BlinkStick's LEDs all show **one color at a time** — per-LED patterns are a documented non-goal (#11, closed as documented).
- **One light at a time**: cues are one-at-a-time on one routed device; SMACC does not fire a BlinkStick and a Hue simultaneously.
- **No flash on Hue**, and Hue onsets lag their markers — see [the comparison](#).

6.8 Troubleshooting

- **The BlinkStick isn't listed** — replug it and press **F5** (or click **Refresh devices** in the Devices window). The driver ships inside SMACC; nothing to install.
- **"No light is set."** — bind the device in the **Devices** window *and* check that *Play visual cue* is routed to that light.
- **The Hue bridge stopped responding** — its IP probably changed; re-enter it in **Set up Philips Hue...** (and give the bridge a DHCP reservation so it stays put). If pairing fails from lab Wi-Fi, see the network note above.
- **A light stayed on** — SMACC turns lights off on every stop and on quit, so a stuck light means the app was killed mid-cue or the device dropped: replug the stick, or power-cycle the bulb (or toggle it in the Hue app).

7. Biocals

Sleep studies open with **biocalibrations**: scripted participant actions (eyes open, eyes closed, look left/right, and so on) whose known physiological signatures verify the recording channels. The **Biocals** window (in the **Panels** column) runs them as timed, marked events — every standard biocal plus the common lucid-dreaming signal practices (LRLR variants, fist clenches, sniffs), each on its own row.

☐ Seq	☒ Voice	Biocal	Duration			
☒	☒	Eyes Open	30 s		▲	▼
☒	☒	Eyes Closed	30 s		▲	▼
☒	☒	Look L/R	15 s		▲	▼
☒	☒	Look U/D	15 s		▲	▼
☒	☒	Blink	10 s		▲	▼
☒	☒	Eyes Closed	30 s		▲	▼
☒	☒	Clench Jaw	10 s		▲	▼
☒	☒	Flex Feet	15 s		▲	▼
☒	☒	Hold Breath	10 s		▲	▼
☒	☒	Breathe	30 s		▲	▼
☒	☒	Rest	60 s		▲	▼

Eyes Open ▼ + Add

7.1 Running a biocal

Each row has a toggle button, two checkboxes, and a duration:

- **The biocal's button** runs it. The button stays depressed while it runs and the countdown at the top shows the time remaining in its task window. Press it again to cancel early, so a botched 30-second window need not be waited out.
- **Voice** speaks the pre-recorded instruction (for example, *"Please close your eyes and relax for thirty seconds."*) over the cue output when the biocal starts. Leave it unchecked to give the instruction yourself.
- **Seq** includes the row when **Play sequence** runs the whole stack in order. Standard biocal's start checked; the lucid-dreaming ones start unchecked.
- **Duration** is the task window in seconds. With the voice on, the window (and its countdown) starts when the instruction *ends*, so a 10-second breath hold is a full 10 seconds. This matches the manual practice of speaking first, then marking when the participant complies.

7.2 Sequences

Play sequence runs every Seq-checked row top to bottom, depressing each button as it goes. Pressing the *active* biocal's button skips that item (a cancel marker, then on to the next); pressing the sequence button again aborts the rest. Rows can **repeat** a biocal (eyes-closed twice, extra LRLRs — use + **Add** for another instance), be reordered with ▲/▼, and removed with ✕. The stack is locked while something is running. For a biocal SMACC does not ship, use a custom event button in the [Event logging panel](#).

7.3 Markers

Each biocal's *start* code fires when its task window opens. A shared **completed** code fires when the window runs out, and a shared **cancelled** code fires on an early stop (the preceding start code identifies which biocal). A played sequence is bracketed by its own start/stop codes and otherwise fires the same per-biocal markers. The defaults are sequence start/stop **105/106**, cancelled **107**, completed **108**, and one start code per biocal in the **110–126** band. All are retunable in the **Markers** window like any built-in event (see [Markers & port codes](#)).

7.4 Voice recordings

Voice recordings ship inside SMACC (generated with [ElevenLabs](#) text-to-speech) and are read straight from the bundle, so they stay current when you upgrade. To use another voice or language, drop your own recording under the same name in your SMACC directory's **biocal's/** folder (for example **~/SMACC/biocal's/**); a file there overrides the bundled one. A biocal with no recording in either place still runs, just unvoiced, and session start warns when that happens.

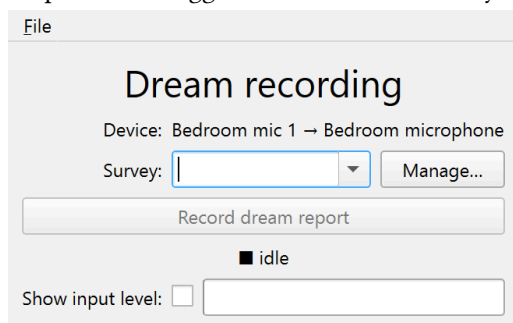
The shared **Voice volume** rides the cue route, so the master output cap and the control-room monitor fan-out apply to instructions exactly as they do to cues (see [Volume & latency](#) and [Audio routing](#)).

8. Dream reports & surveys

After an awakening the participant gives a **dream report**: recorded audio, often followed by a questionnaire. SMACC records the audio and can open a survey alongside it.

8.1 Recording a dream report

Use the **Record dream report** button (in the **Dream recording** panel) to record from the mic bound to the **Bedroom mic 1** equipment in the **Devices** window. Recordings are saved into the current session folder as `report-NN.wav`. Each report is stamped with the time elapsed since you pressed **Start recording** (in the [Event logging panel](#)), so it is easy to locate in the EEG file. If recording has not been marked yet, the report is still logged and SMACC reminds you to mark it.



8.2 Surveys

A dream report is often followed by a questionnaire: how lucid was the dream, what did it contain, how confident is the participant. SMACC handles these two ways:

- **In-app surveys** render in a SMACC window and save their responses straight into the run folder, next to the dream-report WAV they accompany. SMACC ships the standard lucid-dreaming instruments, and you can build your own.
- **Web surveys** (for example a questionnaire hosted on Qualtrics or REDCap) open in the browser; add them by URL.

In-app surveys exist because the night shift is a bad place for a hosted form: sleep labs are often offline, response data belongs with the night's recording rather than a third-party account, and overnight questionnaires are frequently administered *verbally over the intercom*, with the participant in bed in the dark while the experimenter reads the items and records the answers. The survey window is non-modal for the same reason, so the intercom stays reachable while it is open.

8.3 Built-in surveys

SMACC bundles standard instruments from the dream and mindfulness literatures (where a survey's definition carries a citation, it is shown at the top of its window):

Dreaming & lucidity

Survey	Full name
DLQ	Dream Lucidity Questionnaire
LuCiD	Lucidity and Consciousness in Dreams scale
LUSK	Lucid Dreaming Skills Questionnaire
MUSK	Morning Lucid Dreaming Skills Questionnaire

Survey	Full name
--------	-----------

BLA	Baseline Lucidity Assessment
------------	------------------------------

MLA	Morning Lucidity Assessment
------------	-----------------------------

Dream traits / intake

Survey	Full name
--------	-----------

MADRE	Mannheim Dream Questionnaire
--------------	------------------------------

MADRE is a trait/intake questionnaire (dream-recall habits, attitudes, age, occupation), administered once at session setup rather than after each awakening — open it standalone from **File > Surveys**. It mixes response types (dropdowns, number and text entry, and a rating matrix), which is what motivated the typed items described under [Building your own](#).

Mindfulness

Survey	Full name
--------	-----------

CAMSR	Cognitive and Affective Mindfulness Scale - Revised
--------------	---

FFMQ	Five Facet Mindfulness Questionnaire
-------------	--------------------------------------

FMI	Freiburg Mindfulness Inventory
------------	--------------------------------

KIMS	Kentucky Inventory of Mindfulness Skills
-------------	--

MAAS	Mindfulness Attention Awareness Scale
-------------	---------------------------------------

MACE	Metacognition, Affect, Cognitive Experiences Questionnaire
-------------	--

PHLMS	Philadelphia Mindfulness Scale
--------------	--------------------------------

SMAAS	State Mindfulness Attention Awareness Scale
--------------	---

SMS	State Mindfulness Scale
------------	-------------------------

TMS	Toronto Mindfulness Scale
------------	---------------------------

Built-ins ship with SMACC itself (they are not stored in your `.smacc` study file), so updates reach every install. Each survey definition carries a content `version` that is recorded in every response file — an analysis can always tell which wording a given night used.

8.4 Using a survey

The survey dropdown in the **Dream recording** panel lists the in-app surveys first, then your saved web URLs. Pick one to open it automatically when a dream report starts recording; that administration is *attached* to the report, and its response file is named after it. Any survey can also be opened standalone from **File > Surveys**.

Every open logs a `SurveyOpened` event. Submitting an in-app survey writes the response file and logs a `SurveySubmitted` event (code 71) — log-only by default, since the survey happens after the awakening; tick its **LSL/TTL** boxes in the [Markers window](#) if your protocol wants it in the trigger channel.

In the Editor (and the read-only previews in the Manage dialog) a survey renders but cannot be submitted — there is no run folder to save into.

8.5 Response files

One JSON file per administration, in the run folder:

- `report-02-survey-dlq.json` — auto-opened with dream report 2 (sorts beside `report-02.wav`).
- `survey-01-lusk.json` — opened standalone; standalone administrations are numbered in their own sequence.

The payload records the SMACC version, the survey's key, name, title, and content version, the optional subject/session metadata, opened/submitted timestamps, the time since the **Start recording** marker (like the dream-report stamp), the linked report number (or `null`), the shared scale with its anchors, one `{item,`

`type`, `response`, `label`} entry per responding item, and any free-text notes. The `response` is the chosen value (or `null` for unanswered — submitting warns first); `label` is its human reading (the Likert anchor or the chosen dropdown option). Display-only headings collect nothing and are omitted:

```
{
  "kind": "smacc/survey-response",
  "smacc_version": "0.1.0",
  "survey": {"key": "dlq", "name": "DLQ", "version": "1.0", "builtin": true, "...":
  "..."},
  "report_number": 2,
  "time_since_recording_start": "02:14:09",
  "responses": [
    {"item": "...", "type": "likert", "response": 3, "label": "..."}
  ]
}
```

Scoring is deliberately left to analysis: SMACC records raw per-item responses only, so scoring conventions can never silently drift between studies.

8.6 Building your own

The **Manage...** button next to the survey dropdown manages all three kinds of entries: view a built-in, build/edit/remove your own, or add a web URL.

Build survey... creates a custom in-app survey: a name, optional title/citation/instructions, one shared rating scale (its range plus an optional anchor label per point), and the items, one per line. That single-scale shape is deliberate — it matches the bundled instruments and keeps definitions simple. Custom surveys are saved as YAML files in your SMACC folder's `surveys/` directory (`~/SMACC/surveys/` by default) — *not* in the `.smacc` study file — and load alongside the built-ins on every launch. Bump the **Version** field whenever you change the items.

The files are hand-editable (same format as the bundled ones):

```
kind: smacc/survey
schema_version: 1
key: my-survey          # stable id; names the response files
name: MySurvey          # short label shown in the dropdown
title: My Survey's Full Title
version: "1.0"          # recorded in every response
citation: ""
instructions: "Rate each statement about the dream you just reported."
scale:
  min: 0
  max: 4
  anchors: [Not at all, Just a little, Moderately, Pretty much, Very much]
items:
  - I was aware that I was dreaming.
  - ...
```

8.6.1 Item types

A bare-string item is a **Likert** item rated against the survey's shared `scale` — the original shape, and all the builder produces. Definition files can also mix in other item types by writing an item as a mapping with a `type`:

type	Renders as	Extra fields
likert (default)	radio matrix on the shared <code>scale</code>	—
select	dropdown	levels (a <code>value:</code> label mapping)
number	number entry (blank = unanswered)	min, max, unit (all optional)
text	single-line text entry	—

type	Renders as	Extra fields
heading	a bold section title (collects no response)	—

Any item may carry a `help:` line, shown under it (used for the definitions in MADRE). Consecutive Likert items collapse into one shared matrix; other types render as their own rows. The simple **Build survey...** dialog only authors plain Likert surveys; a survey with typed items is edited as a file (the dialog says so and points at the path). For a full worked example, see the bundled `madre.yaml`.

```
kind: smacc/survey
schema_version: 1
key: intake
name: Intake
title: Session intake
scale: {min: 0, max: 4, anchors: [Not at all, "", "", "", Totally]}
items:
  - text: Age
    type: number
    min: 0
    max: 120
    unit: years
  - text: How often do you recall your dreams?
    type: select
    levels: {0: Never, 1: Sometimes, 2: Often}
  - text: Attitude towards dreams      # a section title
    type: heading
  - I think that dreams are meaningful. # a Likert item on the shared scale
```

Web survey URLs, by contrast, are saved in the `.smacc` study file (`survey_options`) and travel with the study — see the [settings-file reference](#).

9. Intercom & chat

The **Intercom** window (in the **Panels** column) is the live channel between the control room and the bedroom. It carries voice both ways, plus a typed text channel for when audio would intrude.

The screenshot shows a software window titled "Intercom" with a "File" menu. Below the title bar, it displays two sections: "Intercom" and "Text chat".

Intercom section:

- Top status: "To participant: Bedroom speaker → Bedroom speakers • mic: Control-room mic → Bedroom microphone"
- Buttons: "Talk (to participant)" and an empty input field.
- Bottom status: "From participant: off • mic: Bedroom mic 1 → Bedroom microphone"
- Buttons: "Listen (to participant)" and an empty input field.

Text chat section:

- A large empty text area for chat messages.
- Below the text area are four buttons: "Please describe everything that was going throu...", "Are you awake?", "Going back to sleep now.", and "Type to the participant...".
- Below these buttons is a text input field with the placeholder "Type to the participant..." and a "Send" button.
- At the bottom are two more buttons: "Pass keyboard to participant" and "Manage quick messages...".

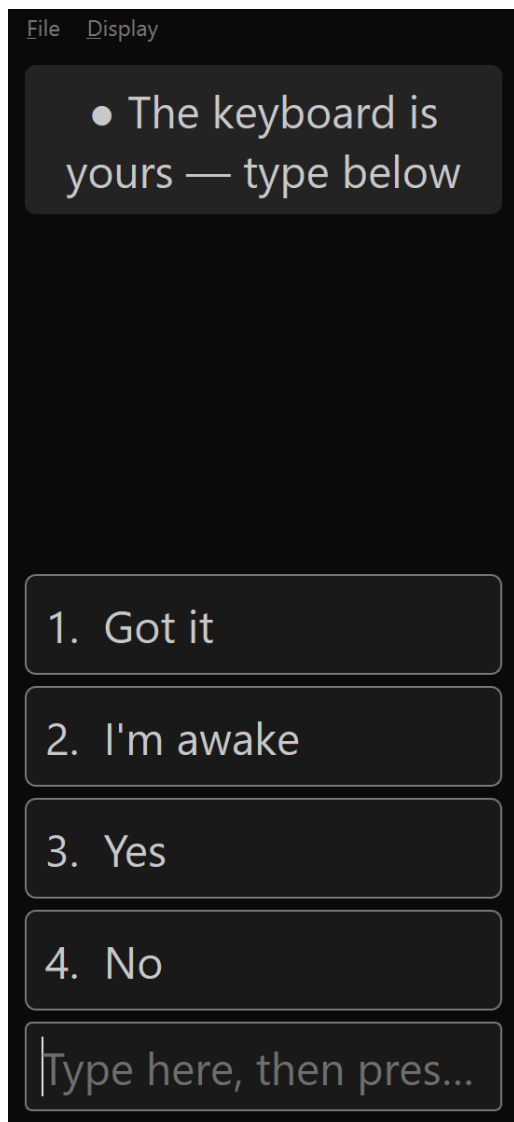
9.1 Voice

- **Talk (to participant)** pipes the control-room mic to the participant's output. Click to latch, or hold the **spacebar** anywhere in SMACC for push-to-talk. Talking is marked in the EEG record.
- **Listen (to participant)** brings the bedroom mic to your control-room speakers. It is unmarked.

Both directions route through equipment set once in the **Devices** window (see [Audio routing](#)). A **level meter** beside each button shows the live input level while that direction is on, so signal on the bar means audio is actually flowing (your mic for Talk, the participant's mic for Listen) rather than just a latched button.

9.2 Text chat

Below the voice controls is a **typed channel**, for hearing-impaired participants, or whenever audio would intrude or you want the exchange in writing. You type in the Intercom panel; the participant reads and replies in a separate **Participant chat** window built for a dark bedroom: always dark regardless of the lights toggle, large text (default 18 pt, resize with **Ctrl+=** / **Ctrl+-**), and an optional **red night text** mode (**Display** menu), with no flashing and no sounds.



Setup. The first cut assumes one computer: extend the desktop onto a bedroom-facing monitor and plug in a second keyboard for the participant. Click **Pass keyboard to participant** to open the participant window, drag it onto the bedroom display once, and it reopens there next session.

One keyboard at a time. Windows gives the machine a single input focus, so the two keyboards type into whichever window is active; text chat is half-duplex, like push-to-talk. **Pass keyboard to participant** (or **Ctrl+Enter**, which sends your message first) activates the participant window so their keystrokes land there. Clicking back into SMACC takes focus back. The participant window shows a banner, “● The keyboard is yours” or “○ Waiting”, so a drowsy participant always knows whether typing will land.

Quick replies. Canned messages save both sides from typing a full sentence at 3 a.m., and they travel with the study. Click **Manage quick messages...** on the Intercom panel to edit two lists: *experimenter prompts* (one click sends a standardized prompt, such as the lab’s dream-report question or “Are you awake?”) and *participant replies* mapped to the number keys 1–9 (for example “Got it”, “I’m awake”, “Yes”, “No”). The participant’s replies appear as large numbered chips; pressing a number sends that reply, but only while their entry box is empty, so a typed reply that contains digits still works. A sent preset is logged and marked exactly like a typed message.

What’s recorded. Every message is written verbatim to the session log as a **DEBUG** line (tick *Debug* above the log preview to watch the exchange live). By default no port codes fire and nothing reaches the BIDS events export: a typed exchange is rapid and conversational, and would flood the marker channel. If a study needs marker timestamps, route **Chat to participant** (code 69) and **Chat from participant** (code 70) to LSL/TTL in the **Markers** window (see [Markers & port codes](#)). The markers stay bare, without the message text, so the trigger channel stays legible.

10. Markers & port codes

This page explains what EEG **triggers** and **port codes** are, the ways SMACC can send them, and how to configure each one. If you just want the steps, jump to [Configuring trigger output in SMACC](#).

10.1 What are port codes and triggers?

When you run a sleep or dream study you want to know *exactly when* things happened — a cue played, REM was observed, a dream report began — and line those moments up with the EEG recording. The standard way to do this is a **trigger**: at the instant of an event, the stimulus computer sends a small number to the EEG amplifier, which records it on a dedicated **trigger channel** alongside the brain signal. That number is the **port code** (also called an event marker or trigger code). During analysis you find events by their code — e.g. “every 41 is an observed REM onset.”

A port code is an **8-bit value**, so it is always an integer from **1 to 255**. That range follows from the hardware: a trigger is physically **eight on/off lines** (eight bits), and the amplifier reads them as one byte. SMACC keeps every code inside 1–255 for this reason. See [Configuring codes](#) for how to view and edit which event sends which code.

10.2 Terminology

The words *event*, *marker*, *trigger*, and *port code* get used loosely in the field; SMACC uses each one for exactly one thing, in the UI, the docs, and the session log:

Term	Meaning in SMACC
------	------------------

Event	A named thing that can happen during a session — a cue started, REM observed, lights off. Each is an entry in the study’s event registry, with a label and a port code.
--------------	---

Marker	The durable record produced when an event fires. A marker is always a log line ; if the event is routed to a transport, it also carries the port code there.
---------------	--

Port code	The 8-bit number (1–255) identifying an event on the amplifier’s trigger channel. Also called a <i>trigger code</i> .
------------------	--

Trigger	The act of <i>sending</i> a port code over a transport. An event can be logged without being triggered.
----------------	---

Transport	A path that carries the code to the recording: the LSL marker stream, or a hardware TTL line (serial trigger box / parallel port).
------------------	--

Log level	The severity tag on a log line (DEBUG...CRITICAL). The log <i>file</i> records every level; levels only filter the live preview. See log levels .
------------------	--

10.3 Default event codes

These are SMACC’s built-in event markers and their default port codes — the out-of-the-box `event_codes` registry. A study can retune any code in the **Markers** window; the change travels in its `.smacc` and is written into every session log.

Code	Event	Key	Notes
41	REM detected	REMDetected	sleep-stage observation (keypad 4 in the Event logging window)
42	Tech in room	TechInRoom	

Code	Event	Key	Notes
43	Training start	TrainingStart	start of a training/learning phase (TMR cue learning, TLR practice)
44	Training end	TrainingEnd	
45	Signal observed	SignalObserved	a lucidity/communication signal; the signal type + confidence are logged as the detail
46	Sleep onset	SleepOnset	
47	Lights off	LightsOff	
48	Lights on	LightsOn	
49	Clapper	Clapper	sync marker
50	Note	Note	
51	Start recording	RecordingStarted	sets the reference clock for dream-report timestamps
52	Wake detected	WakeDetected	sleep-stage observation (keypad 0 in the Event logging window)
53	N1 detected	N1Detected	sleep-stage observation (keypad 1)
54	N2 detected	N2Detected	sleep-stage observation (keypad 2)
55	N3 detected	N3Detected	sleep-stage observation (keypad 3)
56	Arousal detected	ArousalDetected	a brief transient arousal (distinct from a sustained Wake)
57	Artifact detected	ArtifactDetected	EEG artifact (movement, electrode noise, etc.)
60	Cue started	CueStarted	
61	Cue stopped	CueStopped	
62	Noise started	NoiseStarted	
63	Noise stopped	NoiseStopped	
64	Intercom started	IntercomStarted	
65	Intercom stopped	IntercomStopped	
66	Visual started	VisualStarted	
67	Survey opened	SurveyOpened	
68	Visual stopped	VisualStopped	the light is actually off (pairs with 66)
69	Chat to participant	ChatMessageSent	typed intercom message; log-only by default
70	Chat from participant	ChatMessageReceived	participant's typed reply; log-only by default
71	Survey submitted	SurveySubmitted	in-app survey responses saved; log-only by default
100	SMACC initialized	TriggerInitialisat	startup connection test; not a stimulus marker

Code	Event	Key	Notes
105	Biocal sequence started	<code>BiocalSequenceStarted</code>	shared; a played biocal sequence
106	Biocal sequence stopped	<code>BiocalSequenceStopped</code>	shared; completed or aborted
107	Biocal cancelled	<code>BiocalCancelled</code>	shared; the preceding start code identifies the biocal
108	Biocal completed	<code>BiocalCompleted</code>	shared; the task window ran out
110	Biocal: Eyes Open	<code>BiocalEyesOpen</code>	biocal starts mark the task-window opening
111	Biocal: Eyes Closed	<code>BiocalEyesClosed</code>	
112	Biocal: Look L/R	<code>BiocalLookLR</code>	
113	Biocal: Look U/D	<code>BiocalLookUD</code>	
114	Biocal: Blink	<code>BiocalBlink</code>	
115	Biocal: Clench Jaw	<code>BiocalClenchJaw</code>	
116	Biocal: Flex Feet	<code>BiocalFlexFeet</code>	
117	Biocal: Hold Breath	<code>BiocalHoldBreath</code>	
118	Biocal: Breathe	<code>BiocalBreathe</code>	
119	Biocal: Rest	<code>BiocalRest</code>	
120	Biocal: LRLR Open	<code>BiocalLRLROpen</code>	
121	Biocal: LRLR Closed	<code>BiocalLRLRClosed</code>	
122	Biocal: LRLR Slow	<code>BiocalLRLRSlow</code>	
123	Biocal: Fist Clench	<code>BiocalFistClench</code>	
124	Biocal: Fist Closed	<code>BiocalFistClosed</code>	
125	Biocal: Sniff Open	<code>BiocalSniffOpen</code>	
126	Biocal: Sniff Closed	<code>BiocalSniffClosed</code>	
200	Dream report stopped	<code>DreamReportStopped</code>	

Code	Event	Key	Notes
201	Dream report started	DreamReportStart	Increments per report (201, 202, ...)

Codes are integers in **1–255** and must be unique among events routed to a transport (LSL or TTL).

10.4 Configuring codes

Open the **Markers** window from the **Panels** column (in a Session or in the Editor). It is the home for everything about event signaling: a **routing legend** (what the log file, the live preview, LSL, and TTL each receive, and which switch governs it), the full event registry grouped by category (including the events with no grid button — lights, panel controls, biocals, chat, system), and the **hardware TTL transport**. For each event you can set:

- **Code** — the 8-bit port code (1–255) sent when the event triggers.
- **LSL** — whether a firing sends the code over the LSL marker stream.
- **TTL** — whether a firing sends the code over the hardware TTL trigger. The column is grayed out until a transport is enabled in the window’s **Hardware TTL transport** section (the ticks are kept and re-arm with it). An event with neither LSL nor TTL ticked is log-only.
- **Preview** — whether the event shows in the live log preview. The session log *file* always records every event regardless; this only controls the on-screen preview.
- **Increment** — give an event a unique, increasing code on each firing (for example **dream reports**: 201, 202, 203, ...) so individual occurrences are findable in the trigger channel. Off uses one fixed code each time.

TTL safe max code raises a soft warning for TTL-routed codes above it, handy when your trigger hardware accepts only a limited range (some older systems do; LSL carries any code). Codes must be unique among routed events and within 1–255; the window blocks anything else.

Custom events. Use **Add event...** — in the **Event logging** panel itself, or in the Markers window — to create your own button events (a label and a code). They appear in the Event logging panel alongside the built-ins and can be removed again with the Markers window’s **Remove**. Built-in events can be retuned but not removed or renamed.

Edits are staged until you press **Apply** (which validates them first); **Revert** re-reads the session’s current setup. The window stays available throughout a session. If you change a code mid-session, the change is written to the log with a timestamp, so the code-to-event mapping for that session is always recoverable.

Beyond port codes, SMACC logs the important interactions too — volume, colour, device, and fade changes — as plain log lines (no port code), so the session record is complete.

10.5 Event logging panel

The manual event buttons (the sleep-stage family, Signal observed, Sleep onset, Note, your custom events, and so on) live in the **Event logging** panel — open it from the Session window’s **Panels** column. The sleep-stage buttons take a fixed keypad — **0** Wake, **1** N1, **2** N2, **3** N3, **4** REM — and the remaining buttons take **5–9** in order; the shortcuts are active while the panel is focused. The **Lights** toggle stays on the main window (it also flips the dark theme).

Signal observed. One button covers every lucidity/communication signal a study uses (LRLR, sniff, facial, ...), so you do not need a separate button per signal. Pick the **signal** type (the box is editable — type your own and it is remembered for the rest of the session) and a **confidence** (certain / probable / possible) beside the button. Pressing it fires the marker immediately and logs your selection as the detail, so the marker’s timing tracks the observation. Confidence is recorded as a comment; it never changes whether the marker reaches the EEG.

10.6 Where codes live

Your codes are saved in the SMACC file (so they travel with it) and written into every session **.log** (both the initial and final settings blocks), so any session is self-documenting: you can decode its markers later even if the codes changed mid-session.

10.7 How SMACC sends triggers

SMACC emits markers over **LSL** (Lab Streaming Layer), a network marker stream. On top of that, you can *optionally* enable **one** hardware transport so a physical trigger reaches an amplifier that doesn't read LSL. Both fire from the same place, and each event routes to either path independently: its **LSL** and **TTL** flags in the registry decide where its code goes (both by default; an event routed to neither is log-only).

Transport	What it is	What you need
LSL (always on)	A network marker stream recorded by an LSL-aware recorder (e.g. LabRecorder → XDF). Works on the same computer or across the network.	Nothing extra.
Serial (USB trigger box)	The modern replacement for the parallel port. The box appears as a COM port; SMACC writes the code as one byte and the box mirrors it onto 8 TTL lines.	A USB-serial trigger box and its COM port.
Parallel port (LPT)	The classic 25-pin port. Eight data pins carry the code byte, sampled by the amplifier.	An LPT port (often an add-in card) and the InpOut32 driver (see below).

i LSL stays on

Enabling a hardware transport does **not** turn LSL off — events routed to both (the default) reach the LSL stream and the hardware line together. The hardware path is purely additional. LSL support ships inside SMACC itself — there is nothing separate to install on the *sending* side (the recorder, e.g. LabRecorder, is its own program).

💡 Why route per event?

Most studies leave every event on both transports. Per-event routing earns its keep when the TTL hardware is restricted — an older amplifier that only accepts a limited code range can keep its key events on TTL (inside the [safe max](#)) while chattier or higher-coded events still reach the LSL stream.

10.7.1 What is the baud rate?

For the **serial (USB trigger box)** transport you also set a **baud rate** — the speed, in bits per second, at which SMACC talks to the box over its COM port. It is a property of the *serial link*, not of the trigger codes: the same 8-bit code (1–255) is sent either way; baud only controls how fast that byte goes out.

The one rule that matters is that **both ends must use the same baud rate**. Your box is configured (by a switch, firmware, or its manual) to expect a specific rate, and SMACC has to match it — a mismatch produces garbled or missed triggers, not an error.

- **Where to find it:** check your trigger box's manual or its configuration utility. Common rates are 9600, 19200, 38400, 57600, **115200**, and 230400.
- **SMACC's default is 115200**, which many modern USB trigger boxes (e.g. typical BrainProducts/Neurospec-style adapters) use out of the box. If yours specifies a different rate, pick it from the dropdown (or type any other value).
- **Higher isn't "better."** A faster rate shaves only microseconds off a one-byte write, which is negligible next to audio/event timing — so choose the rate your box expects rather than the largest one.

If triggers don't register, a wrong baud rate is one of the first things to check (alongside the COM port and the [pulsed vs. set-and-hold](#) mode).

10.8 Pulsed vs. set-and-hold

Amplifiers and trigger boxes differ in how they expect the code to appear on the lines, so SMACC offers two modes:

- **Pulsed** — SMACC raises the code on the lines, waits a configurable **pulse width** (e.g. 10 ms), then drops them back to 0. Each event is a brief, distinct pulse. Choose this when the amplifier expects a momentary trigger, or when you want SMACC to control the pulse length.
- **Set-and-hold** — SMACC writes the code once and leaves it on the lines until the next event. Choose this for amplifiers that sample a held level, **and** for boxes that generate their own fixed-width pulse when they receive a byte (SMACC just sets the value; the box shapes the pulse).

If you're not sure which your hardware wants, start with **pulsed at 10 ms** and verify with the **Test** button (below); switch to set-and-hold if events don't register cleanly.

10.9 Configuring trigger output in SMACC

1. Open the **Markers** window from the Panels column (available both in a live Session and in the Editor) and find its **Hardware TTL transport** section.
2. Tick **Enable hardware trigger output**.
3. Choose a **Transport**:
 - **Serial** — pick your box's **Port** from the dropdown (click **Refresh** if you plugged it in after opening the window) and set the **Baud** rate to match your box (see [What is the baud rate?](#); SMACC defaults to 115200). If the rig isn't attached right now, you can type the port name (e.g. COM3) directly.
 - **Parallel port** — enter the **Address** as hex (see [Finding your parallel-port address](#)).
4. Choose a **Mode** (pulsed or set-and-hold) and, for pulsed, a **Pulse width**.
5. Click **Test** to send one pulse and confirm the amplifier sees it. The result appears next to the button; an error explains what went wrong.
6. Click **Apply**.

The whole configuration is saved in your [SMACC file](#), so it travels with the rest of your setup. Because a COM port name or LPT address is specific to one computer, SMACC reports a clear error if the saved port can't be opened on the machine you load it on — re-pick it in the Markers window and save again.

 Always validate on the real hardware

A successful **Test** confirms SMACC could open the port and write to it. It does **not** prove the amplifier recorded the correct code on its trigger channel. Before relying on triggers for a study, record a few test events and confirm they appear, with the right codes, in the EEG.

10.10 Finding your parallel-port address

A parallel port is addressed by a base **I/O address**, written in hexadecimal. The most common values are:

- **0x378** — the usual address for the first parallel port (LPT1).
- **0x278** — a common second port (LPT2).
- **0x3BC** — older onboard ports.

Add-in PCIe/PCI parallel cards are frequently mapped somewhere else entirely, so **don't assume** — look it up:

1. Open **Device Manager** (press **Win+X**, then choose Device Manager).
2. Expand **Ports (COM & LPT)** and double-click your parallel port.
3. Go to the **Resources** tab and read the first **I/O Range** — the start of that range is your address. Enter it in SMACC with a **0x** prefix (e.g. 0x378).

10.11 Installing the InpOut32 driver (parallel port only)

Modern Windows blocks direct port access from ordinary programs, so the parallel-port path needs a small kernel driver, **InpOut32** / **inpoutx64**. SMACC does **not** download or install it for you (an earlier version did, and it proved fragile and required admin rights every launch). Install it once, manually:

1. Download **InpOutBinaries** from highrez.co.uk.
2. Run the included **InstallDriver.exe** once (as administrator) to install the kernel driver.
3. Make sure **inpoutx64.dll** is where SMACC can load it — on the system path, in **C:\Windows\System32**, or beside **SMACC.exe**.

If the driver isn't available, SMACC's parallel transport reports a clear error and falls back to LSL only — it never crashes a session over a missing driver.

10.12 Verifying

- Use the **Test** button in the Markers window's transport section for a quick “can SMACC drive the line?” check.
- Then confirm end-to-end on the amplifier: record a short block, fire a few events from the **Event logging** window, and check that the codes land on the EEG trigger channel. Only the real recording proves the path works.

III Devices & routing

11	Audio routing	51
11.1	Equipment and actions	51
11.2	One audio engine	53
11.3	Monitoring and volume	53
11.4	Windows only	54
12	Compatible devices	55
12.1	BlinkStick	55
12.2	Philips Hue	55
13	Volume & latency	57
13.1	Volume	57
13.2	Latency	57
13.3	What contributes	58
13.4	The marker marks the software event, not the photons	58
13.5	Measured: a typical Windows rig	59
13.6	The Low / High setting	59
13.7	Visual cues: BlinkStick vs Hue	59
13.8	Measure it on your own rig	59
13.9	If you want it lower	60
14	EEG Annotator	61
14.1	Supported recordings	61
14.2	Viewing	61
14.3	Annotating	62
14.4	Sleep staging	63
14.5	Session log overlay	63
14.6	Multiple raters	64
14.7	Blind-rater mode	64
14.8	For developers	65
15	Troubleshooting	67
15.1	Common problems	67
15.2	If SMACC crashes	67
15.3	Reporting a problem	67
16	Release notes	69
16.1	0.1.1 — 2026-06-15	69
16.2	0.1.0 — 2026-06-15	69

11. Audio routing

An overnight cueing study can run several audio streams at once: a cue in the bedroom, white noise in the bedroom, your voice over an intercom, the participant's voice coming back, a dream-report mic, and sometimes a [light cue](#). These often span more than one physical device and two rooms. On Windows, the same speaker can also appear under several names, and the level that reaches the participant is the product of several controls spread across the OS.

This page describes how SMACC handles devices and routing. For the output safety cap and stimulus timing, see [Volume & latency](#).

11.1 Equipment and actions

Instead of picking a device separately in every window, SMACC has one **Devices window** (in the **Panels** column) where you do two things:

1. **Bind each piece of equipment to a device, once.** Equipment entries are the physical endpoints of a rig, named by place: **Bedroom speaker**, **Control-room speaker**, **Bedroom mic 1** (and an optional **Bedroom mic 2**), the **Control-room mic** (the experimenter's intercom voice), plus the light devices (**BlinkStick light**, **Philips Hue light**).
2. **Route each action to equipment.** Every action names what SMACC does with a device — *Play audio cue*, *Record dream report* — and each points at a piece of equipment. Hover any row in the window for a full description.

The cue, the noise, and your intercom voice can all share the **Bedroom speaker**, so swapping it is one change rather than several. Every other window shows a read-only indicator of where it resolves, for example `Device: Bedroom speaker → Speakers (USB Audio)`, and the Devices window shows the same resolution beside each route — a route pointed at equipment with nothing bound reads **→ no device** instead of looking configured.

File

Devices

Tell SMACC what equipment the rig has, then which equipment each action uses.
Hover any row for what it does.

Equipment → devices (what the rig has)

Bedroom speaker is:

Control-room speaker is:

Bedroom mic 1 is:

Bedroom mic 2 is:

Control-room mic is:

BlinkStick light is:

Philips Hue light is:

Actions → equipment (what SMACC does with it)

Play audio cue using: → Bedroom speakers

Listen to audio cue using:

Play noise using: → Bedroom speakers

Play visual cue using: → BS000001

Speak to participant using: → Bedroom speakers

Listen to participant using:

Record dream report using: → Bedroom microphone

Monitor bedroom noise using: → Bedroom microphone

Bedroom speaker	Speakers (USB Audio)
Control-room speaker	Headphones (Realtek)
Bedroom mic 1	Microphone (USB Audio)
Play audio cue	→ Bedroom speaker (listen: Control-room speaker)
Play noise	→ Bedroom speaker
Speak to participant	→ Bedroom speaker
Record dream report	→ Bedroom mic 1

The whole assignment is saved in your SMACC file, so a rig travels with its study. If a bound device isn't connected when a study loads, SMACC keeps going and reports which one is missing instead of silently falling back.

11.1.1 No “system default”

SMACC never routes audio to “whatever the system default is”. Windows reassigns its default device on its own — plugging in an HDMI monitor or a Bluetooth headset is enough — and an overnight study must not follow it. Instead:

- When a session starts (or loads a study) with nothing bound to the **Bedroom speaker**, **Bedroom mic 1**, or the **Control-room mic**, SMACC binds the device that is *currently* the Windows default — explicitly,

by name — and logs which one it picked. From then on the binding is pinned: changing the Windows default never re-routes a study.

- Equipment with nothing bound reads **(none)**, and anything routed to it reports a clear error instead of quietly playing somewhere else.
- If no device is connected at all, the dropdown says so (e.g. **No output device found**) rather than offering an empty choice.
- A bound device that isn't currently connected shows as **(not connected)** and is kept — never silently swapped — until you pick another device or plug it back in (then **Refresh devices (F5)**).

The study editor never auto-binds: a study built on an office machine arrives at the rig with its equipment unbound, and the rig pins its own defaults on first load.

11.1.2 Monitoring routes

Three optional routes cover the things you reach for mid-study:

- **Listen to audio cue (fan-out)**. Route *Listen to audio cue* to the control-room speaker and each cue plays in the bedroom and the control room at once, so you hear what the participant hears.
- **Listen to participant**. The intercom is two-way: **Speak to participant** sends your voice — picked up by the **Control-room mic** — to the participant (and is marked in the EEG record), while **Listen to participant** brings the participant's mic to your control-room speaker. The intercom also has a typed [text-chat mode](#) (no audio device involved) for hearing-impaired participants.
- **Monitor bedroom noise**. A microphone meter in the Audio cue window that confirms a cue is audible in the bedroom (see [Audio cues](#)). It defaults to **Bedroom mic 1**, or bind a dedicated, more sensitive **Bedroom mic 2** and route *Monitor bedroom noise* to it.

11.2 One audio engine

All real-time audio (cues, noise, the intercom, the dream-report recorder, and the input-level meter) runs on `sounddevice` (PortAudio), through the **Windows WASAPI** host API.

i Why WASAPI

On Windows, PortAudio lists every speaker once per host API (MME, DirectSound, WASAPI, WDM-KS), which is where the “same device, several names” confusion comes from, and the legacy MME names are truncated to 31 characters. SMACC enumerates only WASAPI, so each device appears once, with its full name, on the modern low-latency path.

i Device names in the Devices window

Because SMACC enumerates only WASAPI devices, it drops the redundant “, Windows WASAPI” suffix that PortAudio appends to each device name. With only one host API in the list, the suffix added nothing, so a device reads as **Speakers (USB Audio)** rather than **Speakers (USB Audio), Windows WASAPI**.

Because everything shares one engine, device names mean the same thing everywhere, and sending one cue to two devices at once (the fan-out) is two streams from a single source.

Hot-plug: plug a device in after SMACC is already open and it's picked up automatically, with no restart needed. You can also force a rescan with the **Refresh devices (F5)** button in the Devices window.

11.3 Monitoring and volume

To confirm a cue actually reaches the bedroom, use the Audio cue window's **Monitoring** meters — see [Audio cues](#). For the output safety cap, the per-cue gains, and how the Windows volume stages combine, see [Volume & latency](#).

11.4 Windows only

This device and volume handling is Windows-specific (WASAPI, the volume mixer), and the GUI toolkit (Qt 6) requires Windows 10 or later.

12. Compatible devices

SMACC can drive a small number of external devices for cueing. This page lists the supported hardware and what each one needs. For how SMACC *assigns and routes* devices — equipment, the Devices window, and volume — see [Audio routing](#); for using the light devices — patterns, timing, choosing between them, and safety — see [Visual cues](#).

12.1 BlinkStick

[BlinkStick](#) is a USB-controlled RGB LED device. SMACC uses it for **visual cues** — lighting up a chosen color, steady or pulsing/flashing, for a set duration — which is handy for light-based cueing in sleep and lucid-dreaming experiments.

12.1.1 What you need

- **A BlinkStick device.** They are sold directly from [blinkstick.com](#). SMACC drives them through the [blinkstick](#) Python library, so any model that library supports (BlinkStick Square, Strip, Nano, Flex, ...) will work.
- **Nothing else to install.** The BlinkStick driver is bundled inside `SMACC.exe`, so you only need to plug the device into a USB port before launching SMACC.

12.1.2 Using it in SMACC

1. Plug the BlinkStick into a USB port, then launch SMACC.
2. Click **Visual cue** in the **Panels** column.
3. Bind your BlinkStick to the **BlinkStick light** equipment in the **Devices** window (in the **Panels** column). Plug one in after launch and it's detected automatically, or click **Refresh devices (F5)** in the Devices window to rescan.
4. Configure a light cue — color, brightness, pattern (steady, or a pulse/flash at a rate in Hz), and length (or **Loop** until stopped) — and add more cues with + **Add cue** if the protocol needs several. See [Visual cues](#) for what each pattern is for.
5. Click a cue's **Play** to fire it; **Stop** turns the light off early. The rest of SMACC stays responsive while the light is on.

Your chosen device and the whole cue board are saved in the [SMACC file](#), so the visual-cue setup travels with the rest of your configuration; the device is reconnected by serial on the next launch (and flagged if it isn't plugged in).

Note

If the visual cue window reports that no light is set, open the **Devices** window and bind one to **BlinkStick light** (plug it in first; click **Refresh devices (F5)** there if it isn't listed). No restart needed.

12.2 Philips Hue

A [Philips Hue](#) bridge is the room-scale alternative to the BlinkStick: the visual cue drives an ordinary Hue bulb (or a whole group) over the local network. Setup is once per bridge:

1. In the **Devices** window, click **Set up Philips Hue...**
2. Click **Find bridge** (or type the bridge's IP — the Hue app shows it under bridge settings).
3. Press the round **link button** on the bridge itself, then click **Pair** within 30 seconds. **Test** lists the bridge's lights to confirm.

4. Bind a light or group to the **Philips Hue light** equipment, and route **Play visual cue** to **Philips Hue light**.

The bridge IP and pairing key are stored in the study's `.smacc` (the key is a local-network credential — treat the file accordingly). Hue suits ambient, room-scale light; it can't flash and its onsets lag their markers, so time-locked protocols should keep the BlinkStick — see the full [BlinkStick-vs-Hue comparison](#), and the network note there if the rig lives on a university Wi-Fi.

13. Volume & latency

The **Volume** window holds two things that shape every cue: how loud it can get (the output safety cap and the visible OS volume stages) and the output **latency** setting. This page covers both.

13.1 Volume

On Windows, the level reaching the participant is a product of several controls:

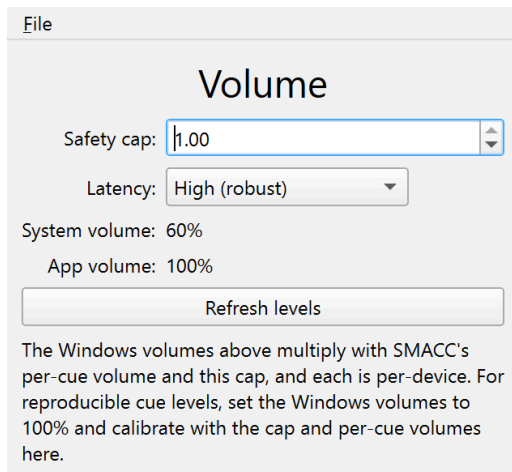
```
per-cue volume × output safety cap × Windows app volume × Windows device volume  
× hardware knob
```

Three of those live in the OS and are invisible from most apps. SMACC makes its own gain explicit and adds a safety limit, in the **Volume** window:

- **Output safety cap.** A single master ceiling, applied as the last gain stage on every cue and noise output. However loud an individual cue is set, the cap is a hard limit, so a full-volume looped cue on a calibrated rig cannot suddenly blast a sleeping participant.
- **A read-only view of the Windows stages.** The window shows the current **System volume** (the Windows output endpoint) and **App volume** (SMACC's own level) in the Windows Volume Mixer, so the hidden OS stages are visible.

💡 Calibrating cue level

For levels that reproduce across nights and participants, set the Windows device and app volumes to 100%, then calibrate with the per-cue volumes and the safety cap inside SMACC. That way one place, SMACC, determines how loud a cue is.



13.2 Latency

How long is it from clicking **Play** (or firing a cue) to the participant actually hearing the sound or seeing the light, and how well does the event marker line up with it? This section answers that, shows how to measure it on your own rig, and explains what SMACC does to keep markers honest.

The short version

For the dream-engineering work SMACC is built for — cueing for lucidity or memory reactivation — **absolute latency is rarely critical**. A cue landing 20 ms early or late doesn't change whether it reaches REM. What matters is that the **marker lines up with the stimulus in the EEG**, and there a *constant* offset is harmless (you can correct it after the fact) while **jitter** — random variation — is not. SMACC therefore optimises for *predictable* timing, not minimal latency.

13.3 What contributes

From the click to the sound, in order:

Stage	Typ- ical	Notes
Click → handler	< 1 ms	Qt event dispatch.
Decode + resample	a few ms	The cue is decoded when loaded; only the resample runs at play time.
Open the output stream	tens of ms	A fresh WASAPI stream is opened per cue (see #105). Affects <i>click→sound</i> , not the marker.
Output buffer	~3–25 ms	The stream's buffer: sound reaches the DAC about one buffer after it starts. This is the marker↔sound gap.
DAC → speaker → air	~1–5 ms	Hardware + ~3 ms per metre of air.

The two that matter are separate axes with separate fixes:

- **Marker → sound** (the science): the **output buffer**. SMACC corrects for this (below), and it's what a per-rig measurement pins down.
- **Click → sound** (operator feel): dominated by the per-play **stream open**. Not a marker-alignment problem; tracked separately in #105.

13.4 The marker marks the software event, not the photons

By default an event marker is timestamped when SMACC *fires* it — which is **before** the sound leaves the speaker (by one output buffer) or, for a light, around when the device write completes. To keep the marker aligned with the stimulus:

- **Audio cue / noise**. SMACC stamps the marker (its LSL timestamp **and** the canonical log line) at the *estimated onset* — the moment it fires **plus the stream's reported output latency** — so the marker tracks the sound rather than SMACC's buffer, and stays put when you change the latency setting. The raw software-trigger instant is kept on a **DEBUG** line in the log for audit.
- **Visual cue**. The first frame is written to the device **synchronously**, and the marker fires right after, so it trails the photons by microseconds (BlinkStick) rather than leading them.

This is an estimate, not a measurement

The correction uses the latency PortAudio *reports*. The residual — fade-in ramp, DAC, speaker, the Hue bridge — is small and roughly constant, but only a recording of the actual onset pins it down. If you need sample-accurate alignment, measure your rig (below) and apply the leftover offset; that per-session capability is tracked in #104.

13.5 Measured: a typical Windows rig

Run the bundled probe from the project (it needs no special hardware for this part):

```
uv run tools/measure_latency.py
```

On a stock laptop (Realtek WASAPI shared-mode output) it reports:

```
Device:      Speakers (Realtek(R) Audio)
Sample rate: 48000 Hz
Reported:    low  3.0 ms   high 10.0 ms
Negotiated [low]: 22.0 ms (SMACC stamps the marker this far ahead of the stream start)
Negotiated [high]: 22.0 ms (SMACC stamps the marker this far ahead of the stream start)
```

Two things to take from this:

1. The **negotiated** latency (~22 ms) is what you actually get — *not* the smaller “reported” figure. SMACC corrects the cue marker by this negotiated value.
2. On **shared-mode WASAPI**, the engine rounds up to its own period, so **Low and High come out the same** here. The Low setting only helps on devices/drivers whose buffer it can actually shrink; true low latency needs WASAPI *exclusive* mode, which SMACC doesn’t use today. **Don’t assume Low helps — measure it.**

13.6 The Low / High setting

The Volume window has a **Latency** choice (High / Low), saved in the `.smacc (output_latency)`:

- **High** (default) — PortAudio’s robust buffer. Fewer underruns; the safe choice for an overnight looping cue, where a glitch in a sleeper’s ear is worse than 20 ms.
- **Low** — asks for a smaller buffer. Trims the marker↔sound gap *where the device allows it* (often no change on shared-mode WASAPI). It applies to the next cue or noise played, not the one currently sounding.

Because the marker is corrected by the *negotiated* latency, switching High/Low never silently shifts your markers — the correction tracks the setting.

13.7 Visual cues: BlinkStick vs Hue

Light latency depends entirely on the backend:

- **BlinkStick** — a USB-HID write, a few milliseconds. Fast enough that the marker effectively coincides with the light.
- **Philips Hue** — each command is an HTTP call to the bridge (~100 ms) plus the bridge→bulb hop, and the bridge rate-limits, so Hue is unsuitable for tight timing or flashing (SMACC refuses `flash` on Hue for this reason). Fine for slow ambient cues; don’t time-lock analyses to a Hue marker without measuring it.

13.8 Measure it on your own rig

The numbers above are specific to one machine — **don’t quote them for a different rig**. To characterise yours:

- **Round-trip, on the bench.** Couple the output back to an input (a loopback cable, or a microphone at the speaker) and run:

```
uv run tools/measure_latency.py --loopback --repeats 30
```

It plays tone bursts, hears them back, and reports the distribution. Watch the **jitter** (the spread), not just the mean.

- **Against the EEG, per session (authoritative).** Record the real stimulus onset on a spare amplifier channel — a microphone for the cue, a photodiode for the light — alongside the LSL port code. The

difference between the port code and the recorded onset, across the night, is the true distribution. SMACC does not plot this itself yet; a per-session view of it needs a spare channel and is tracked in [#104](#).

i Why the log can't give you this

You might hope to read latency straight from the log, since it records both the raw trigger time ([DEBUG](#)) and the corrected onset ([INFO](#)). But their difference is just the latency SMACC *added* — a constant, not a measurement. A real distribution needs an independent observation of the stimulus, i.e. the methods above.

13.9 If you want it lower

In rough order of effort: prefer **short, sharp-onset cues** (a long fade-in is its own latency); try the **Low** setting and measure whether your device honours it; keep the machine unloaded at night. A persistent pre-warmed stream ([#105](#)) and WASAPI exclusive mode would cut more, but neither is a SMACC goal today — for lucidity and TMR cueing, predictable beats fast.

14. EEG Annotator

The **EEG Annotator** is SMACC's post-hoc viewer for recorded EEG: open a file, scroll through the night, apply display filters, and place named annotations — saved as a small sidecar file next to the recording, which is **never modified**. It is a review tool, not a real-time display; nothing about it runs during a live session.

It opens from the Launcher's **EEG Annotator** button and always runs as its own window and process — so you can keep reviewing last night's file while tonight's session runs.

How it differs from a generic EEG viewer. The Annotator is built for the overnight cueing workflow, not for clinical reading:

- It is **read-only by contract** — the recording is never written; every mark, stage score, and overlay lives in a small sidecar beside it.
- It is **SMACC-aware** — it imports SMACC's own portcodes off the recording and can overlay a night's [session log](#) (cues, dream reports, even the recorded report audio), so you see *what SMACC did* on the same timeline.
- It has **blind, multi-rater scoring** built in (see [blind-rater mode](#)) — for objective signal and stage scoring, not an afterthought.
- Its **epoch model** is decoupled from the on-screen window and can be anchored to any feature.
- Recordings are **memory-mapped**, so an 8-hour high-density night opens in seconds and scrolls smoothly regardless of file size.

i Built in, runs in its own process

The Annotator is part of SMACC — there is nothing extra to install. It opens in its own process, so it can outlive the Launcher and a heavy recording can never disturb a running session; that is why it appears as a separate window rather than a panel inside the session app.

14.1 Supported recordings

Format	Open via
European Data Format	<code>.edf</code>
BrainVision	<code>.vhdr</code> (of the <code>.vhdr/.eeg/.vmrk</code> triplet)
FIF (MNE / Elekta)	<code>.fif</code>
Neuroscan	<code>.cnt</code>
EEGLAB	<code>.set</code> (continuous recordings only)

The open dialog's file filter is generated from this list, so it never drifts from what the tool can actually read. An *epoched* `.set` is not a continuous recording and is rejected with MNE's own message. Recordings are memory-mapped, never loaded whole: an 8-hour high-density night opens in seconds and scrolling stays smooth regardless of file size.

14.2 Viewing

- **Window length** — 10/30/60/120 s pages; **30 s** is the default. This is the *on-screen* window, separate from the [scoring epoch](#).
- **Filters** — high-pass, low-pass, and a 50/60 Hz notch, applied to the *display only* (zero-phase, so nothing shifts in time). Recordings open **unfiltered** — the usual sleep view is two clicks away (HP 0.3 Hz, LP 35 Hz).

- **Scale** — microvolts per channel lane; smaller numbers mean bigger traces. Trigger/stim channels are auto-fit to their lane.
- **Channels...** picks which channels are shown and reorders them; each channel type (EEG, EOG, EMG, ...) scales on its own. **Save profile...** stores the whole montage — channels, filters, scale, and the window/epoch lengths — to a file, and **Load profile...** applies it to any recording, so a lab's house view is one click on every night.
- **Export figure...** writes the current window to a publication-ready **PNG, PDF, or SVG**.
- The status bar shows the cursor's time from recording start **and the wall-clock time**, so events line up with the night's session log.

14.2.1 Keyboard navigation

The arrow keys drive the view from anywhere in the window — you do **not** have to click the traces first:

Key	Action
← / →	step back / forward one epoch
Shift+← / →	nudge 1 s, to peek across a boundary
↑ / ↓	bigger / smaller traces (Shift = fine)
Home / End	jump to the start / end of the recording
PageUp / PageDown	page back / forward one window

The mouse wheel and scrollbar scroll as well.

14.2.2 The epoch model

The **scoring epoch** is a first-class concept, deliberately *separate* from the on-screen window — a 30 s epoch can be inspected inside a 60 s window, and the arrow keys step by epoch regardless of how much is on screen.

- **Epoch** length is set in seconds (default **30 s**, the polysomnography standard; 1–300 s).
- **Epoch grid** draws faint, **numbered** epoch boundaries over the traces.
- **Anchor epochs to view** starts an epoch at the left edge of the current view and back/front-fills the whole grid from there (boundaries fall at **anchor + k·epoch**) — so you can line the grid up with lights-out or any feature in the record. **Reset anchor** puts epoch 1 back at the recording start.
- **Time axis** labels the x-axis as wall-clock **Clock** time or **Elapsed** seconds from the start. Clock needs a recording start time; an anonymized file that has none falls back to elapsed.

14.3 Annotating

1. **Drag** across the traces to mark a span (drag never pans — the time axis only moves when you ask it to). To drop a **point** instead, **Ctrl+click** (or press **M**) at the cursor.
2. Name it in the label dialog — recent labels and common marks (LRLR, arousal, artifact, cue response) are one click; free text always works. Tick **instantaneous** to keep just the moment instead of the dragged span.
3. A plain **click** on an annotation selects it; use the side list to rename, delete, or jump to one (double-click).
4. **Save annotations** writes the sidecar next to the recording: `night1.edf` → `night1.annotations.tsv` (+ a small `.json` describing it). The recording itself is **never modified**. Unsaved changes star the title and prompt before closing, and saving over a sidecar **this review did not open** asks first — so you never clobber another file by accident.

For fast signal scoring, the **Quick marks** row drops a labeled point mark at the cursor in one click — or press **1–9** for the first nine. Use **Edit palette...** to set the buttons; the palette defaults to the lucid eye-signal vocabulary (LRLR, LRLRx2, LRLRx3, IEIE).

Autosave & recovery. A couple of seconds after you stop editing, your work autosaves to a distinct `night1.annotations.autosave.tsv` (a per-rater file when a [rater id](#) is set). It is purely a crash net — if the tool finds one when a recording opens, a non-modal banner offers to **Restore** or **Dismiss** it; nothing is ever applied silently, and your real sidecar is untouched until you Save.

Events already stored **inside** the recording — amp markers, SMACC’s own trigger codes — are imported automatically the *first* time a file is reviewed, so your cues appear on the traces alongside your new marks. Once a sidecar exists it is the single source of truth (nothing is re-imported or duplicated).

The sidecar format is documented in the [annotations file reference](#).

14.4 Sleep staging

Score the night a 30 s epoch at a time. Staging is a separate layer from annotating — both are always live on the same view, so you can drop an LRLR or arousal mark mid-sweep without leaving your place — and it saves to its **own** sidecar (`night1.edf` → `night1.stages.tsv`), a partition of one stage per epoch, never mixed into the event annotations.

Stage focus. Press **Tab** (or the **Stage focus** button) to start a sweep. The view locks to one epoch per screen and the epoch grid/length is fixed (so a score can’t land in the wrong span), a **STAGE FOCUS** chip shows in the status bar, and the epoch at the left edge — the one you’re about to score — gets a teal bracket.

Scoring. With stage focus on, the number keys score the epoch at the left edge and **auto-advance** to the next one:

Key	AASM stage
W	Wake
1 / 2 / 3	N1 / N2 / N3
R	REM
0 / Backspace	clear (un-score, stays put)

The stage **buttons** do the same with the mouse and also work outside stage focus (so the number keys stay free for [quick marks](#) until you deliberately start a sweep). Re-scoring an epoch overwrites it; **Left/Right** page back and forth to fix one.

Reading the night. Each scored epoch tints the traces a faint stage colour, the big readout names the current epoch’s stage, and a **hypnogram overview strip** under the traces shows the whole night as a colour staircase — click it to jump anywhere. The status bar counts scored / total epochs.

Scoring manual. AASM (W/N1/N2/N3/R) is the default; switch the **Manual** dropdown to **Rechtschaffen & Kales** (S1–S4 + a Movement-Time epoch) before you start — it locks once the first epoch is scored, and a resumed hypnogram reopens under the manual it was scored with. A sub-epoch **movement or artifact** isn’t a stage under AASM; mark it as an [annotation](#) over the epoch’s stage instead (**Artifact** is a default quick mark).

Each [rater](#) scores to their own `night1.stages.<id>.tsv`, so two scorers of one night produce two hypnograms to compare.

14.5 Session log overlay

Load the night’s SMACC session `.log` onto the timeline as a **read-only reference track** — every marker, cue, dream report, and survey shown where it happened, so you see what SMACC *did* alongside the EEG. Click **Load session log...** in the **Session log** panel. Ticks appear in a thin lane across the top, coloured by log level, with the full message on hover. The per-level checkboxes show or hide levels just like the live preview — **DEBUG** (raw-trigger and volume-edit noise) is off by default. The log is reference context only: it is never editable and never saved into your annotations.

You can load a log **with or without a recording open**. With one, it overlays and aligns to *that* recording’s clock. With none, it opens **standalone** on a bare time axis — for inspecting a log on its own (the **Analyzer’s** *Open log in EEG Annotator* hands a session off this way). Opening a recording while a standalone log is shown switches to the overlaid view.

Artifacts. Selecting a dream-report entry lets you **Play** its recorded audio (`report-NN.wav`, beside the log in the session folder) — press again to stop — and **Reveal file** opens that folder in the file browser. Playback is disabled for an entry with no audio (a marker, or a log opened away from its recordings).

Aligning the log. The log’s timestamps come from the recording PC while the EEG’s clock comes from the amplifier, so the two can differ by seconds or more. Slide the whole log along the EEG to line them up, three ways — all adjusting one offset:

- **Drag the log lane.** A left-drag that starts in the top lane slides the log (a drag anywhere else still draws an annotation). On release it snaps onto a nearby mark of yours, so a clapper lands exactly on the artifact it made.
- **Pair an entry to a feature.** Select a log entry, click **Align entry to feature...**, then click the EEG feature it produced — useful when the two are far apart to drag. (Esc cancels.)
- **Nudge the offset.** The **Offset** box is the exact value, and the fine control.

Auto-align to triggers. When the amplifier itself recorded SMACC’s trigger codes — a hardware-TTL rig, where the codes are embedded events in the EEG file — **Auto-align to triggers** estimates the offset by matching the log’s markers to them. It anchors only on *rare* codes (a clapper, a dream report, the recording-start marker), never the periodic cue codes that could line up at the wrong firing, and grades the result: a confident match is applied silently, a low-confidence one is applied but marked *unverified* in the status bar, and an unreliable or contradictory one (too few matches, or two clusters from a clock jump or the wrong log) is refused so you align by hand instead. It runs automatically when a log is loaded, and the button re-runs it. An LSL-only rig records markers only to its side file, not the EEG, so there is nothing embedded to match — use the manual gestures.

The classic clapper workflow fits directly: press **Clapper** on the SMACC PC at the same instant as a manual trigger on the recording system, then pair (or drag) the logged clapper onto the marker it left in the EEG. Clapping at both lights-off and lights-on gives two reference points. A recording with no start time (anonymized) has no absolute anchor, so the log starts at its first entry and you align it entirely by hand.

The log is **never shown in a blind review** — it records every cue and portcode, which would unblind the rater.

14.6 Multiple raters

For blind, multi-rater scoring, give each reviewer a **rater id** — click the **Rater** button, or launch with `--rater <id>`. Their annotations then save to a per-rater sidecar (`night1.annotations.alice.tsv`) instead of the plain one, so several raters can score the same recording without overwriting each other. The active id shows in the window title and on the Rater button, and the first save under an id confirms it (so a forgotten id is caught). Leave it blank for an ordinary single-rater review. See the [annotations file reference](#).

Once several raters have scored a recording, opening it shows the **other raters** as a read-only overlay — each rater’s marks in their own colour behind your editable ones — with a show/hide checkbox per rater in the **Other raters** list. Only your own marks are clickable and editable; the overlay is for comparison. (Overlays are off during a blind review, so a blind rater never sees their peers.)

14.7 Blind-rater mode

Objective scoring asks raters to judge the EEG without seeing what was already marked. The **Blind** button (or `--blind`) filters annotations **before they are ever shown**, so a rater cannot glimpse what is hidden, with three presets:

- **Fully naive** — every mark is hidden; the rater scrolls a clean recording.
- **Reports visible** — only dream-report markers are shown; detected signals and cues are hidden.
- **Signal-present (classify only)** — signal *positions* are shown with their labels blanked (a ?), so the rater sees *where* a signal is and classifies *what* it is.

Blind mode **requires a rater id** — a blind review seeds from the coordinator’s truth sidecar (`night1.annotations.tsv`) but saves to the rater’s own (`night1.annotations.alice.tsv`), so the truth file is never overwritten and each rater’s judgements stay separate for later comparison. Reopening a rater’s own file resumes their work unfiltered.

A coordinator can save a preset (plus the visible/signal label lists and a quick-mark palette) as a shareable `study.smacc-blind.json` and hand out a one-click command:

```
SMACC.exe --eeg --rater alice --blind study.smacc-blind.json night1.edf
```

Blinding is a workflow aid, not a security boundary: the guarantee is that the app never *renders* a hidden label. Someone with file access can still read the recording or the truth sidecar directly.

14.8 For developers

Run it from a source checkout:

```
uv sync --extra dev
uv run python -m smacc.eeg [recording.edf]
```


15. Troubleshooting

When something goes wrong, start here. Most fixes live on the relevant tool's page; this page collects the cross-cutting ones and points to the rest.

15.1 Common problems

Symptom	Where to look
Windows SmartScreen blocks the download or installer	Installation — expected for the unsigned build; More info → Run anyway
“No output/input device found”, or a bound device shows (not connected)	Audio routing — rebind in the Devices window and Refresh devices (F5)
A cue plays in the control room but maybe not the bedroom	Audio cues — read the <i>Sending</i> and <i>Bedroom</i> meters
The BlinkStick isn't listed, or a light won't fire or stays on	Visual cues
The Hue bridge stopped responding	Visual cues — its IP probably changed
Triggers don't register on the amplifier	Markers & port codes — check the COM port, baud rate, and pulsed-vs-set-and-hold mode
A cue is too loud or too quiet	Volume & latency — the per-cue volume, the safety cap, and the OS stages

15.2 If SMACC crashes

Separately from the per-run logs, SMACC keeps a permanent crash log at `~/SMACC/logs/crash.log`. Uncaught errors, Qt's own fatal messages, and — via Python's `faulthandler` — the stack of every thread at a hard crash (such as an access violation inside Qt or an audio driver) are appended there, even when no session is running. A hard-crash dump shows where the *Python* side was at that moment, which is usually enough to identify the failing subsystem. Every launch and every session start is also stamped in the file, so a crash can be matched to its night and run folder.

The file is rotated at launch once it grows large (one older `crash.log.1` generation is kept). When SMACC crashes it shows a dialog with an **Open logs folder** button that takes you straight to it.

15.3 Reporting a problem

Open an issue on the [SMACC issue tracker](#) with what you did, what happened, and the relevant files: `crash.log` together with the affected run's folder, or a session `.log`.

16. Release notes

⚠️ SMACC is pre-1.0

SMACC has not reached a stable release. Settings files, marker codes, the interface, and behavior may change between releases, and there are no compatibility shims. Pin one version for the duration of a study. See the [versioning policy](#) for what the numbers mean.

16.1 0.1.1 — 2026-06-15

- Fixed the Analyzer vanishing on open — clicking **Analyzer** hid the launcher but never showed the Analyzer window.

16.2 0.1.0 — 2026-06-15

The first published release of SMACC — a Windows control surface for running sleep and dream studies. It includes:

- **Audio cues**, designed in-app with the Audio Cue Designer, plus masking background noise.
- **Visual cues** on a BlinkStick or Philips Hue light.
- **Biocalibrations** as timed, marked tasks.
- **Dream reports and surveys** for structured night-time data collection.
- **Intercom** to talk, listen, and type with the participant.
- **EEG markers** (port codes) over an LSL marker stream or a hardware TTL trigger.
- The **EEG Annotator** for reviewing and scoring recordings (optional component).
- A detailed **session log**, a **volume safety cap**, and explicit **device routing** so a misclick can't blast or mislead a sleeping participant.
- A per-user **Windows installer** (no admin rights) and a portable **SMACC.exe**.

Earlier 0.0.x builds predate this release and are not covered here.

IV

Reference

17	File formats	73
17.1	Where each file lives	73
17.2	Stability promise	73
18	SMACC file (.smacc)	75
18.1	On-disk shape	75
18.2	Envelope	76
18.3	settings	76
18.4	Paths and portability	79
18.5	Version history	79
19	Preferences file (preferences.yaml)	81
19.1	Example	81
19.2	Fields	81
19.3	Version history	82
20	Session log (.log)	83
20.1	Line format	83
20.2	Log levels	83
20.3	Embedded settings blocks	84
21	BIDS export (events.tsv + sidecar)	85
21.1	events.tsv	85
21.2	JSON sidecar	85
22	Annotations sidecar	87
22.1	The TSV	87
22.2	The JSON	87
22.3	Multiple raters	88
22.4	Precedence	88
23	Contributing	89
23.1	Conventions	89
23.2	Commit and pull-request style	89
23.3	Development	90
23.4	Building the executable	90
23.5	Building the docs	91
23.6	Versioning	91
23.7	Project notes	92

17. File formats

SMACC reads and writes a handful of files. This section is the **reference** for each one — its on-disk shape, every field, and how it is versioned — as distinct from the task-oriented [Overview](#) guide.

File	What it is	For- mat	Version
SMACC file (<code>.smacc</code>)	A portable study configuration	YAML	<code>schema_version: 1</code>
<code>preferences.yaml</code>	Per-machine operator preferences	YAML	<code>schema_version: 1</code>
Session <code>.log</code>	The per-run record (events + settings)	Text	—
BIDS export	<code>events.tsv</code> + JSON sidecar	TSV / JSON	follows BIDS
Survey definition	An in-app survey (built-in or custom)	YAML	<code>schema_version: 1</code>
Survey response	One administration's answers	JSON	—
Annotations sidecar	EEG Annotator marks (<code>*.annotations.tsv</code>)	TSV / JSON	follows BIDS columns

17.1 Where each file lives

- The **SMACC directory** (`$SMACC_DIRECTORY`, else `~/SMACC`) holds `preferences.yaml`, the seeded `default.smacc`, and the default data directory.
- A `.smacc` can live anywhere; it names the **data directory** its runs are written to.
- Each **run** gets its own timestamped folder (`smacc-YYYYmmdd-HHMMSS/`) under that data directory, holding the session `.log`, any dream-report audio, survey responses, and exports.
- **Custom survey definitions** live in the SMACC directory's `surveys/` folder (created when you first build or save one); built-in ones ship inside SMACC itself.
- **Bundled assets refresh on upgrade.** `default.smacc` (a read-only template) and the `demo- cues` (seeded into the data directory's `cues/` folder) are re-seeded from the bundle when they change, so a newer SMACC's improvements reach an existing directory; your own files are untouched. Biocal voice recordings are read straight from the bundle, with the SMACC directory's `biocal/` folder as an optional per-recording override.

17.2 Stability promise

`.smacc` and `preferences.yaml` each carry an integer `schema_version`, currently **1** — the first stable release schema. A `.smacc` loads only at the matching version (any other is rejected); `preferences.yaml` never blocks startup — it ignores the version and merges the keys it recognizes over the defaults. Either way, missing *optional* keys fall back to a default, so a partial or hand-edited file still loads.

The `.smacc` format will not change incompatibly without **bumping `schema_version`** and adding a row to its version-history table. The `kind` discriminator (`smacc/settings`, `smacc/preferences`, `smacc/survey`, `smacc/survey-response`) lets SMACC reject a file that isn't its own: a `.smacc` with a *different* kind is rejected, while a missing kind is tolerated.

18. SMACC file (.smacc)

A **SMACC file** is a **portable study configuration**: it lets a researcher set SMACC up once — cue sounds and volumes, noise, the visual cue, survey selection, event codes, device routing, optional hardware triggers — and reload it each session so the setup stays consistent across nights and researchers. It is plain YAML you can read and edit. The user-facing extension is `.smacc`, but the `kind` stays `smacc/settings`.

See [SMACC files](#) for the task-level guide (creating, editing, sharing). This page is the field reference.

18.1 On-disk shape

```
# SMACC settings — YAML (.smacc). Edit with care.
kind: smacc/settings
schema_version: 1
smacc_version: "0.1.0"
metadata:
  subject: "001"
  session: "1"
  notes: ""
  created: "2026-06-09T22:14:00"
settings:
  # --- Audio cues -----
  cues:
    - {name: "Piano cue", file: "cues/piano.wav", volume: 0.4, loop: false}
  cue_attack: 0.0
  cue_release: 0.0
  # --- Background noise -----
  noise_volume: 0.2
  noise_color: white
  noise_source: builtin
  noise_file: ""
  # --- Visual cues (BlinkStick) -----
  visual_cues:
    - {name: "Light 1", color: "#ff0000", brightness: 1.0, pattern: steady, rate: 1.0,
length: 1.0, loop: false}
  visual_attack: 0.0
  visual_release: 0.0
  # --- Survey -----
  survey_url: ""
  survey_options: {}
  # --- Participant text chat -----
  chat_font_size: 18
  chat_red_text: false
  chat_experimenter_presets: ["Please describe everything that was going through your
mind before the alarm.", "Are you awake?", "Going back to sleep now."]
  chat_participant_presets: ["Got it", "I'm awake", "Yes", "No"]
  # --- Biocals -----
  biocals:
    voice_volume: 0.5
    rows:
      - {biocal: eyes_open, sequence: true, voice: true, duration: 30}
  # --- Output cap + latency -----
  volume_cap: 1.0
  output_latency: high
  # --- Device equipment + routing -----
```

```

devices:
  bindings: {}
  routing:
    play_audio_cue: bedroom_speaker
    play_noise: bedroom_speaker
    record_dream_report: bedroom_mic_1
    play_visual_cue: blinkstick_light
# --- Event-marker registry -----
event_codes:
  - {key: REMDetected, code: 41, lsl: true, ttl: true, preview: true, increment: false}
event_code_safe_max: 255
# --- Optional hardware trigger output -----
trigger_output:
  enabled: false
  transport: serial
  port: ""
  baud: 115200
  address: "0x378"
  mode: pulsed
  pulse_ms: 10
# --- Philips Hue bridge (visual cues) -----
hue:
  bridge_ip: ""
  app_key: ""
# --- Data directory + interface -----
data_directory: data
preview_levels: [INFO, WARNING, ERROR, CRITICAL]
always_on_top: false
tool_always_on_top: {}

```

18.2 Envelope

Key	Type	Meaning
kind	string	Always smacc/settings; a file with a different kind is rejected.
schema_version	integer	The settings schema version — currently 1. Only the current version loads; any other is rejected (see Version history).
smacc_version	string	The SMACC version that wrote the file (informational).
metadata	map	Optional run metadata: <code>subject</code> , <code>session</code> , <code>notes</code> , and <code>created</code> (ISO timestamp). Blank by default; recorded in the log, not baked into filenames.
settings	map	The configuration itself (below).
	ping	

18.3 settings

Every panel contributes its own keys; a few window-level blocks travel alongside. Any key may be omitted — each falls back to its default.

18.3.1 Audio, noise, visual, survey, chat, volume

Key	Type	Meaning
cues	list	One entry per cue slot: <code>name</code> (string), <code>file</code> (WAV path), <code>volume</code> (0–1), <code>loop</code> (bool).

Key	Type	Meaning
<code>cue_attack</code>	seconds	Fade-in applied to a starting cue.
<code>cue_release</code>	seconds	Fade-out applied to a stopping cue.
<code>noise_volume</code>	0–1	Background-noise level.
<code>noise_color</code>	string	Selected noise colour (as offered in the Noise panel, e.g. <code>white</code>).
<code>noise_source</code>	multi- file	Generated noise, or a WAV file.
<code>noise_file</code>	path	The noise WAV when <code>noise_source: file</code> .
<code>visual_cue</code>	list	One entry per light slot: <code>name</code> (string), <code>color</code> (<code>#rrggbb</code>), <code>brightness</code> (0–1), <code>pattern</code> (<code>steady</code> <code>pulse</code> <code>flash</code>), <code>rate</code> (Hz, the pulse/flash speed), <code>length</code> (seconds; ignored while <code>loop</code>), <code>loop</code> (bool).
<code>visual_attack</code>	seconds	Brightness fade-in applied to a starting visual cue.
<code>visual_release</code>	seconds	Brightness fade-out applied to a stopping visual cue.
<code>survey_url</code>	string	The selected survey: a web URL, or <code>smacc://survey/<key></code> for an in-app survey.
<code>survey_options</code>	map ping	Named <i>web</i> survey presets: label → URL. In-app surveys (built-in or custom) are not persisted here — they come from survey definition files (see Dream reports & surveys).
<code>chat_font_size</code>	size te- ger	Participant chat window text size, in points (8–72).
<code>chat_red_text</code>	boolean	Red-shifted night text in the participant chat window.
<code>chat_experimenter_prompt</code>	list	Inter-press quick-reply prompts the experimenter sends with one click (verbatim, like a typed message). Omitted → seeded defaults; an empty list is respected.
<code>chat_participant_replies</code>	list	Participant quick replies, shown as numbered chips and sent with the number keys 1–9 (max 9). Omitted → seeded defaults; an empty list is respected.
<code>volume_cap</code>	0–1	Master output safety cap multiplied into every stimulus (<code>1.0</code> = no cap).
<code>output_latency</code>	high low	Output buffer for the cue + noise streams: <code>high</code> is robust (default), <code>low</code> trims marker-to-sound delay where the device allows it (often unchanged on shared-mode WASAPI). See Volume & latency .

18.3.2 biocals

The Biocals window’s stack (see [Biocals](#)): `voice_volume` (0–1, the shared instruction volume) plus one `rows` entry per stack row, in display order. Rows may repeat a biocal (e.g. eyes-closed twice in the played sequence). A missing block — or a block without `rows` — loads the default stack; an empty `rows` list is respected as a deliberately cleared one.

Field	Type	Meaning
<code>biocal</code>	string	Stable biocal id (e.g. <code>eyes_open</code>); unknown ids are dropped.
<code>sequence</code>	boolean	Include this row when the sequence is played.
<code>voice</code>	boolean	Speak the pre-recorded instruction when the biocal starts.
<code>duration</code>	in- te- ger	Task-window length in seconds (1–600); the countdown and the completion marker run on it.

18.3.3 Window-level

Key	Type	Meaning
<code>data_directory</code>	path	Where this study's runs are written (relative to the <code>.smacc</code> , or absolute).
<code>preview_levels</code>	list	Log levels shown in the live preview, e.g. <code>[INFO, WARNING, ERROR, CRITICAL]</code> .
<code>always_on_top</code>	boolean	Whether the main session window floats on top.
<code>tool_always_on_top</code>	map-ping	Per-tool always-on-top, keyed by panel key.

18.3.4 event_codes

The editable event-marker registry: a list of overrides applied over the [built-in registry](#). A study that omits it uses the built-ins; one that overrides a few codes keeps the rest.

Field	Type	Meaning
<code>key</code>	string	Stable event id (e.g. <code>REMDetected</code>).
<code>code</code>	integer	Port code, 1–255.
<code>lsl</code>	boolean	Whether a firing pushes the code over the LSL marker stream.
<code>ttd</code>	boolean	Whether a firing pushes the code over the hardware TTL trigger (when one is configured).
<code>preview</code>	boolean	Whether it shows in the live preview (the file always records it).
<code>increment</code>	boolean	Whether successive firings advance the code (e.g. dream reports).

A built-in entry persists only those fields. A **custom** event also carries `label`, `category`, `tooltip`, and `builtin: false` so it can be reconstructed on load. `event_code_safe_max` (integer, default 255) sets the soft upper-bound warning for TTL-routed codes (LSL carries any code).

18.3.5 devices

Equipment → device bindings plus action → equipment routing (see [Audio routing](#) and [Devices](#)).

Key	Type	Meaning
<code>bindings</code>	map	Equipment key → device key. Equipment: <code>bedroom_speaker</code> , <code>control_speaker</code> , <code>bedroom_mic_1</code> , <code>bedroom_mic_2</code> , <code>control_mic</code> (the experimenter's intercom mic), <code>blinkstick_light</code> (a stick's serial), <code>philips_hue_light</code> (a bridge target like <code>light:3</code> or <code>group:1</code>).
<code>routing</code>	map	Action key → equipment key (" " = off). Actions: <code>play_audio_cue</code> , <code>listen_audio_cue</code> , <code>play_noise</code> , <code>play_visual_cue</code> , <code>speak_to_participant</code> , <code>listen_to_participant</code> , <code>record_dream_report</code> , <code>monitor_bedroom_noise</code> .

A missing `devices` block loads the defaults (each action on its default equipment, with no devices bound). When a live session starts (or loads a study) with `bedroom_speaker`, `bedroom_mic_1`, or `control_mic` unbound, SMACC binds the current Windows default device explicitly, by name, and logs the choice — there is no “system default” pseudo-selection (see [Audio routing](#)).

18.3.6 trigger_output

Optional hardware TTL trigger output, mirrored alongside the always-on LSL stream (see [Markers & port codes](#)).

Field	Type	Meaning
<code>enabled</code>	boolean	Whether the hardware path is on (LSL is always on regardless).
<code>transport</code>	<code>serial</code> <code>parallel</code>	USB trigger box, or parallel (LPT) port.
<code>port</code>	string	Serial COM port, e.g. <code>COM3</code> .
<code>baud</code>	integer	Serial baud rate (default 115200).
<code>address</code>	string	Parallel-port base address as hex, e.g. <code>0x378</code> .
<code>mode</code>	<code>pulsed</code> <code>hold</code>	Pulse the code then drop, or set-and-hold until the next event.
<code>pulse_ms</code>	integer	Pulse width in ms when <code>mode</code> : <code>pulsed</code> (default 10).

18.3.7 hue

The Philips Hue bridge used for visual cues (see [Devices > Philips Hue](#)), paired once in the Devices window. Like the device bindings, this is rig state that travels with the study — note that `app_key` is the bridge credential minted by pairing, stored in plain text.

Field	Type	Meaning
<code>bridge_ip</code>	string	The bridge's IP on the rig's network.
<code>app_key</code>	string	The app key minted by press-button pairing (blank = not set up).

18.4 Paths and portability

Cue/noise WAVs and `data_directory` are stored **relative** to the `.smacc` when they sit beside it (POSIX separators) and **absolute** otherwise, so a self-contained study folder stays valid when moved, copied, or zipped.

18.5 Version history

Changes sion

1 Introduced as the v0.0.9 baseline (numbering restarted; pre-release schemas were retired without migration); still current as of v0.1.0. Envelope (`kind` / `schema_version` / `smacc_version` / `metadata` / `settings`); panel state; the `biocals`, `devices`, `event_codes` + `event_code_safe_max`, `trigger_output`, `data_directory`, `preview_levels`, `always_on_top`, and `tool_always_on_top` blocks.

19. Preferences file (`preferences.yaml`)

`preferences.yaml` holds **per-machine operator preferences** — where each window was last placed, the Launcher’s recent-files list, the live log-preview options, and the EEG Annotator’s per-machine state (recent labels, last-used folders, rater id, quick-mark palette). It is the machine layer, kept separate from a portable [SMACC file](#) (which a researcher shares between rigs) and from a per-run [session log](#).

It lives in the SMACC directory (`$SMACC_DIRECTORY`, else `~/SMACC/preferences.yaml`), is loaded at startup, and is written on quit. It must never break the app: a missing or corrupt file falls back to the built-in defaults, and saving swallows errors.

19.1 Example

```
kind: smacc/preferences
schema_version: 1
preferences:
  windows:
    launcher: {x: 100, y: 100, w: 340, h: 360}
    main: {x: 120, y: 80, w: 900, h: 700}
  recent_settings:
    - C:\Users\you\SMACC\peter.smacc
    - C:\Users\you\SMACC\paul.smacc
  last_settings: C:\Users\you\SMACC\peter.smacc
  log_preview_max_lines: 1000
  log_preview_clock: 24h
  eeg_palette_labels: [LRLR, LRLRx2, LRLRx3, IEIE]
```

19.2 Fields

Key	Type	Meaning
<code>kind</code>	string	Always <code>smacc/preferences</code> ; a file with a different <code>kind</code> is ignored (defaults are used).
<code>schema_version</code>	integer	The preferences schema version (currently 1).
<code>preferences</code>	mapping	The preferences themselves (below).

19.2.1 `preferences`

Key	Type	Meaning
<code>windows</code>	mapping	Per-window geometry, keyed by a stable window id $\rightarrow \{x, y, w, h\}$. Ids include <code>launcher</code> , <code>pingmain</code> (the Session window), <code>analyze</code> (the Analyzer), <code>eeg-annotator</code> (the EEG Annotator), and one entry per tool window (keyed by its panel key). An absent/ <code>null</code> <code>x/y</code> means “no saved position — open at a default”.
<code>recent_settings</code>	list	Recently opened <code>.smacc</code> files, most-recent first, de-duplicated and capped at 8.
<code>list_recently_of_paths</code>	list	of paths

Key	Type	Meaning
last_session_path	path or null	The last .smacc opened, used to preselect it in the Session.../Editor... file picker.
log_preview_lines	int	How many lines the Session window's live log preview keeps (default 1000); the oldest lines are dropped first. The log <i>file</i> always records everything, so nothing is lost. Very large values cost GUI memory and repaint time over an overnight session.
log_preview_clock	string or null	How the live preview renders the time of day: 24h (default, e.g. 22:14:01) or 12h (10:14:01 PM). Presentation only — the log <i>file</i> always keeps 24-hour timestamps with a UTC offset. Toggle it from the Session window's File → 12-hour clock ; an unknown value falls back to 24h.

i Partial files are fine

Loading merges a file's keys over the defaults, so a file missing some keys still yields every key. There is no cross-version migration; only `schema_version: 1` is current.

19.2.2 EEG Annotator keys

The **EEG Annotator** runs as its own process but writes its per-machine state into the same `preferences.yaml`:

Key	Type	Meaning
eeg_recent_labels	list	Recent annotation labels, seeding the label dialog.
eeg_rater_id	string or null	The active rater id for per-rater sidecars.
eeg_palette_labels	list	Quick-mark palette (default [LRLR, LRLRx2, LRLRx3, IEIE]).
eeg_last_dir	path or null	Last folder a recording was opened from.
eeg_last_profile_dir	path or null	Last view-profile folder.
eeg_last_export_dir	path or null	Last figure-export folder.
eeg_last_blind_dir	path or null	Last blind-config folder.

19.3 Version history

Version	Changes
1	First (and current) stable schema, covering window geometry, recents/last-used, the log-preview options, and the EEG Annotator keys. (The pre-release <code>association_prompted</code> key was dropped along with the first-run association prompt — the installer owns the association now; a leftover key in an old file is ignored.)

20. Session log (.log)

Every live run writes one plain-text log to its own timestamped folder (`smacc-YYYYmdd-HHMMSS/`) under the study's data directory. It is the session's record: every event marker, the soft interactions (volume / colour / device changes), and two embedded snapshots of the full settings the run used. The **Editor** (which records nothing) writes no log.

20.1 Line format

Each line is three comma-separated fields:

```
YYYY-MM-DD HH:MM:SS.mmm±HHMM, LEVEL, message
```

- **timestamp** — local wall-clock, millisecond precision, with the machine's UTC offset (e.g. `-0500`). The offset lets a reader place the night on an absolute timeline — for example when overlaying the log on an EEG recording whose clock sits in another zone. Logs written before SMACC recorded the offset are *timezone-naive* (no `±HHMM`); both forms are read back the same way. The *file* is always 24-hour; the live on-screen preview can optionally show 12-hour (AM/PM) time (Session window → **File** → **12-hour clock**), which changes only the display, not what is written.
- **LEVEL** — a Python logging level: `DEBUG`, `INFO`, `WARNING`, `ERROR`, or `CRITICAL`. The file records every level; the live on-screen preview shows only a configurable subset.
- **message** — the log text. An **event-marker** line ends in " - portcode N":

```
2026-06-09 22:14:01.003-0500, INFO, Opened SMACC v0.1.0
2026-06-09 22:14:05.221-0500, INFO, Lights off - portcode 47
2026-06-09 22:18:30.880-0500, INFO, Dream report started: report-01, t+00:04:29 -
portcode 201
2026-06-09 22:19:02.114-0500, INFO, REM detected - portcode 41
```

A marker line is "`{label} - portcode {code}`" when the event drives a trigger, or just "`{label}`" when it does not. A dream-report start names its recording (`report-NN`, matching `report-NN.wav` in the run folder) and, once the recording-start marker has been set, its time since that marker, so the entry can be tied back to both its audio and its place in the EEG. The code-to-event map is the study's `event_codes` registry; see the [default code catalog](#).

20.2 Log levels

The file records **every** level — a level never decides whether something is written, only whether it shows in the live preview (whose default gate starts at `INFO`). SMACC assigns levels by one convention:

Level	What it carries
-------	-----------------

<code>DEBUG</code>	Housekeeping and high-frequency detail: settings loads/saves, device rescans, live volume edits, raw trigger instants, chat text. In the file for the record; out of the preview by default.
--------------------	--

<code>INFO</code>	Event markers and meaningful operator actions — the session's scientific narrative.
-------------------	--

<code>WARNING</code>	Mid-session configuration changes (a port code or trigger transport edited during a run — loud so the code map stays traceable) and recoverable faults (a saved device not connected).
----------------------	--

<code>ERROR</code>	Faults that cost something: a hardware trigger write failing, a stream that couldn't open.
--------------------	--

<code>CRITICAL</code>	Caught exceptions — the app is in an unknown state.
-----------------------	---

20.2.1 Stimulus marker timing

Most markers are stamped when SMACC fires them. **Audio cue and noise** markers are the exception: their timestamp — in the log line *and* the LSL stream — is the *estimated onset* (the fire time plus the output stream's reported latency), so the marker lines up with the sound rather than SMACC's buffer (see [Volume & latency](#)). The raw software-trigger instant rides alongside on a `DEBUG` line:

```
2026-06-09 22:18:30.858-0500, DEBUG, Cue started: Piano cue: software trigger at
22:18:30.858, marker advanced +22.0 ms to estimated onset (output latency)
2026-06-09 22:18:30.880-0500, INFO, Cue started: Piano cue - portcode 60
```

That `DEBUG` line is deliberately **not** a " - portcode N" line, so the [BIDS export](#) counts the event once, at its onset.

20.2.2 Text-chat transcript

Each [text-chat](#) message is written verbatim to a `DEBUG` line, one per message — in the file for the record, out of the live preview and the BIDS export by default:

```
2026-06-09 23:41:12.402-0500, DEBUG, Chat to participant: Are you comfortable?
2026-06-09 23:41:35.118-0500, DEBUG, Chat from participant: yes
```

If a study flips the chat events' triggers on, the marker lines fire alongside — bare (`Chat to participant - portcode 69`), without the message text, so the trigger channel and the export stay legible.

20.3 Embedded settings blocks

The log carries the **complete settings the run used**, so a session stays self-documenting even if the study file later changes. The block is the same payload as a [.smacc file](#), but every line is prefixed with `#` and fenced by sentinels, so log parsers skip it entirely:

```
# --8<-- smacc/settings initial
# kind: smacc/settings
# schema_version: 1
# smacc_version: 0.1.0
# metadata:
#   subject: '001'
#   ...
# settings:
#   ...
# --8<-- end smacc/settings initial
```

Two snapshots are written: **initial** (at startup) and **final** (appended at quit). The **final** block may be absent if a session crashed before quitting. The **Analyzer** can recover a `.smacc` from either block.

i No separate version

The log itself isn't versioned; its embedded blocks carry the [settings](#) `schema_version`.

21. BIDS export (events.tsv + sidecar)

SMACC can convert a session [log](#) into a [BIDS](#) events file — a tab-separated `events.tsv` plus a JSON sidecar describing its columns. Export one from the **Analyzer** with its **Export events (BIDS)**... button. Only event-marker lines (" - portcode N") become rows.

21.1 events.tsv

Tab-separated, one header row plus one row per marker:

```
onset    duration    trial_type    value
5.5 n/a Lights off 47
1810.25 n/a REM detected 41
```

Column	Type	Meaning
onset	seconds	Time from the first parseable log line (the onset origin).
duration	seconds or n/a	n/a for instantaneous markers (all SMACC markers are instantaneous).
trial_type	string	The event label, exactly as logged.
value	integer	The event's port code (1–255).

21.2 JSON sidecar

Written alongside as `<name>.json`, describing each column. `value` points back to the log's `event_codes` block for the full code-to-event map of that session:

```
{
  "onset": {
    "Description": "Event onset relative to the first log entry.",
    "Units": "second"
  },
  "duration": {
    "Description": "Event duration; 'n/a' for instantaneous markers.",
    "Units": "second"
  },
  "trial_type": {
    "Description": "Event label as logged by SMACC."
  },
  "value": {
    "Description": "SMACC event-marker port code. The full code-to-event map for the session is recorded in its .log settings block (under event_codes)."
  }
}
```


22. Annotations sidecar

The [EEG Annotator](#) saves annotations as a **TSV + JSON sidecar pair** next to the source recording, which is never modified:

```
night1.edf
night1.annotations.tsv    ← the annotations
night1.annotations.json   ← column documentation + provenance
```

The pair is named by replacing the recording's extension with `.annotations.tsv` / `.annotations.json`. It is deliberately **not** BIDS's `_events.tsv` name: opening a recording that lives inside a BIDS dataset must never clobber the dataset's own events file. The columns themselves follow the BIDS/MNE convention, so downstream tooling reads them with no surprises.

22.1 The TSV

Tab-separated, UTF-8, one header row, sorted by onset:

```
onset    duration    description
12.345   0.000    LRLR
80.500   22.000   Arousal
```

Column	Meaning
<code>onset</code>	Seconds from the start of the recording's data (not clock time), millisecond precision
<code>duration</code>	Seconds; <code>0.000</code> marks an instantaneous event
<code>description</code>	The label as entered by the reviewer

Overlapping annotations are allowed (an arousal inside a REM period is two rows). A label containing a double quote is csv-quoted (`"saw a "light""`); tabs/newlines never appear inside fields — labels are whitespace-normalized when created.

The file is hand-editable: the reader accepts Windows line endings and a UTF-8 BOM (e.g. a Notepad re-save), but it is strict about the header and the column count, so a corrupted file is reported instead of silently losing rows.

22.2 The JSON

Documents each column (BIDS-style) and records provenance:

```
{
  "onset": {"Description": "Annotation onset relative to the start of the recording's data.", "Units": "second"},
  "duration": {"Description": "Annotation duration; 0 for an instantaneous mark.", "Units": "second"},
  "description": {"Description": "Annotation label as entered by the reviewer."},
  "SourceFile": "night1.edf",
  "MeasurementDate": "2026-06-05T22:00:00+00:00",
  "Rater": null,
  "GeneratedBy": {"Name": "SMACC", "Version": "0.1.0"}
}
```

`MeasurementDate` is the recording's start as stored in the file — combined with the data-relative onsets it reconstructs clock time. It is `null` when the format or anonymization dropped it. `Rater` names the reviewer for a per-rater sidecar (see below) and is `null` for an ordinary single-rater review. `GeneratedBy.Version`

is the SMACC app version that wrote the file; the sidecar carries no schema version of its own (it tracks the BIDS columns).

22.3 Multiple raters

Objective signal scoring often uses several reviewers. Give each a **rater id** — from the Rater button, or by launching with `--rater <id>` — and that reviewer's annotations save to a sidecar keyed by the id, so several raters can score one recording without overwriting each other:

```
night1.annotations.tsv      ← single-rater review (no id)
night1.annotations.alice.tsv ← rater "alice"
night1.annotations.bob.tsv  ← rater "bob"
```

Each rater writes a different path, so cross-rater clobbering is impossible and the coordinator's marks in the plain `night1.annotations.tsv` stay untouched. The id is reduced to a filesystem-safe token (letters, digits, dash, underscore), and each rater gets their own crash-recovery autosave (`night1.annotations.alice.autosave.tsv`). Downstream analysis attributes and compares marks by reading the JSON `Rater` field.

22.4 Precedence

When a recording is opened, an existing sidecar **wins**: it is loaded as-is, and the events embedded inside the recording are *not* (re)imported — they were already captured the first time the file was reviewed. Delete or rename the sidecar to start a review from scratch.

23. Contributing

i For both human and AI contributors

This page is the canonical development guide for SMACC. It is written for both human contributors and AI coding assistants — AI agents are pointed here from `AGENTS.md`, so the instructions live here once rather than being duplicated across files.

i Requesting a change? Open a GitHub issue

Human contributors should open a [GitHub issue](#) for new feature requests or bug reports before starting work, so changes can be discussed first.

23.1 Conventions

- Always use `uv` to run Python scripts and install dependencies. Never `pip install` or run naked `python`.
- Use the marker vocabulary consistently in UI text, docs, and docstrings — *event*, *marker*, *port code*, *trigger*, *transport* each mean exactly one thing; see the [terminology table](#).
- Write the docs as **reference**, not marketing: lead with the fact, keep pages scannable (short paragraphs, tables, steps), and go easy on em-dashes and the “not X, but Y” construction. Page filenames are stable, so cross-link with relative links and matching heading anchors.
- Pick log levels by the [session-log convention](#): `DEBUG` for housekeeping/high-frequency detail, `INFO` for markers and meaningful operator actions, `WARNING` for mid-session config changes and recoverable faults, `ERROR` for faults that cost something. The file records every level, so demoting a line to `DEBUG` only moves it out of the default live preview, never out of the record.

23.2 Commit and pull-request style

Keep the history skimmable and merge commits clean.

Commits

- One line only — a subject, with no body or extended description.
- Start with a [Conventional Commits](#) prefix: `feat:`, `fix:`, `docs:`, `refactor:`, `test:`, `chore:`, `build:`, `ci:`, or `perf:`.
- Imperative mood, lower-case after the prefix, no trailing period, and ideally under ~72 characters.
- No AI attribution or co-author trailers.

```
feat: optional hardware TTL trigger output
docs: document audio routing
fix: clamp incrementing port codes to 255
```

Pull requests

- The title follows the same one-line Conventional-Commits rule — on a squash merge it becomes the commit subject (GitHub appends the `(#NN)` PR number).
- The body is a brief summary, not an exhaustive change list: what changed, why, and how it was verified. A few sentences or bullets is plenty.
- No AI attribution footer.

Merging

- Squash-merge, and clear the auto-generated commit body so the merged commit is the one-line title alone — no bundled description or commit list.

23.3 Development

```
uv sync --extra dev          # create the environment with dev tools
uv run python -m smacc       # launch the app
uv run pytest                # run the test suite
uv run ruff check .          # lint
uv run ruff format .         # format
uv run mypy                  # type-check
uv run --extra docs mdformat . # format Markdown (docs, README, ...)
pre-commit install           # enable the lint/format/type-check git hooks
```

`uv run smacc` also works once the environment is synced — it’s the installed GUI launcher and opens no console window, so prefer `python -m smacc` when you want terminal output (e.g. tracebacks).

23.3.1 Tests

The suite runs headless. `tests/conftest.py` selects Qt’s `offscreen` platform before any Qt import, so the GUI tests (built on `pytest-qt`) construct windows and panels with no display and no popups — there’s no extra setup on your part. Hardware access (audio device enumeration, the Windows volume read-out) is stubbed in fixtures, so tests don’t depend on the machine’s audio setup.

```
uv run --extra dev pytest --cov=smacc --cov-report=term-missing # CI also adds --cov-report=xml
```

To actually watch a test render, override the platform for that run (Windows: `$env:QT_QPA_PLATFORM = "windows"` before `uv run pytest`).

23.4 Building the executable

Build the Windows app — one `--onedir` bundle that includes the EEG Annotator:

```
uv run pyinstaller entry.py --name SMACC --onedir --noconsole \
  --icon src/smacc/assets/icon.ico \
  --add-data "src/smacc/assets/icon.png:smacc/assets" \
  --add-data "src/smacc/assets/default.smacc:smacc/assets" \
  --add-data "src/smacc/assets/cues:smacc/assets/cues" \
  --add-data "src/smacc/assets/biocal:smacc/assets/biocal" \
  --add-data "src/smacc/assets/surveys:smacc/assets/surveys" \
  --collect-submodules mne --collect-data mne \
  --collect-submodules matplotlib.backends
```

This produces `dist/SMACC/SMACC.exe` (a folder, not a single file). `--onedir` rather than `--onefile` is deliberate: a session launch imports lazily off the install folder and never loads the heavy MNE stack, with no per-launch temp extraction (and so no ~120 MB blob for antivirus to rescan each run). On Windows the `--add-data` separator is `;` rather than `:` (i.e. `...png;smacc/assets`). `--icon` sets the executable’s file icon; `--add-data` bundles the runtime window/taskbar icon, which SMACC resolves via `sys._MEIPASS`.

The EEG Annotator (#136) is a mode of this same binary — `SMACC.exe --eeg`, re-exec’d as its own process (see `smacc.eeg.launch`) — so it is built in, not a second exe. The `--collect-*` flags are what make its MNE stack survive freezing: MNE uses `lazy_loader`, which imports submodules by name at runtime and reads the package’s `.pyi` stubs — both invisible to PyInstaller’s static analysis — so without them the build succeeds and dies on first MNE use. `matplotlib` is a real dependency too: figure export (`eeg/export.py`, #180) writes PDF/SVG via `savefig`, and MNE’s IO layer imports `mne.viz.ui_events` (→ `matplotlib`) at import time — so `--collect-submodules matplotlib.backends` bundles its PDF/SVG backends, and `--exclude-module matplotlib` would break both `read_raw.*` and the export.

Verify the built app with `SMACC.exe --eeg --selftest` (the check is exit code 0): it round-trips a synthetic recording through MNE, the display filters, and the annotation sidecar. A plain `--version` would not catch broken MNE bundling — MNE is imported lazily, so it only loads when a recording is touched.

PyYAML (settings export/import) is pure Python and is picked up automatically by PyInstaller; if a frozen build ever fails to import `yaml`, add `--hidden-import yaml`.

The `blinkstick` driver (BlinkStick visual cues) and its Windows backend `pywinusb` are pure Python and bundle the same way. PyInstaller may warn that `usb.core` is missing — that is BlinkStick’s non-

Windows backend, which SMACC never uses, so the warning is harmless. If a frozen build ever fails to import the driver, add `--hidden-import pywinusb`.

Releases are built automatically: pushing a `v*` tag (e.g. `v0.1.0`) triggers the [release workflow](#), which checks the tag against `smacc.__version__`, builds the onedir `SMACC.exe` (stamped with version metadata from `tools/make_versionfile.py`), wraps it in an Inno Setup installer (`tools/smacc.iss` → `SMACC-Setup.exe`), smoke-tests the app (`--version` and `--eeg --selftest`) and the installer, and attaches `SMACC-Setup.exe` to the GitHub Release. The installer’s asset name is a stable contract — the docs’ download button links `releases/latest/download/SMACC-Setup.exe` — so don’t rename it. The installer’s `[Registry]` section must mirror `winassoc.association_entries()` exactly (so an installed build sees the association as already registered); `tests/test_winassoc.py` cross-checks them.

23.5 Building the docs

The documentation site and the single-file PDF manual are both built with [Quarto](#) — a standalone binary that renders the HTML site and a native Typst PDF from the same Markdown sources under `docs/`, with no plugin stack and no system libraries. Install Quarto once (it bundles the Typst PDF engine); it is not a Python package, so it is not part of `uv sync`.

```
quarto preview docs # live-reload HTML preview while writing
quarto render docs  # build the HTML site and the PDF into docs/_book/
```

The project config is `docs/_quarto.yml` — a Quarto *book*, the project type that yields one combined PDF. Pages are plain Markdown; admonitions are [callouts](#) (`::: {.callout-note}`), and each screenshot carries a `width=` attribute with its description in `fig-alt`. Heading slugs use `gfm_auto_identifiers` so the cross-page `#anchor` links resolve the same way GitHub renders them.

The [docs workflow](#) renders the docs on every pull request and, on pushes to `main`, publishes the site (with the PDF manual) to GitHub Pages. The site is unversioned — one current copy; the manual for an older release is the PDF attached to that release.

Two checks run in CI after the render — run them locally before a docs PR. Together they replace what `mkdocs build --strict` used to cover:

```
uv run python scripts/check_links.py docs/_book #
internal links + heading anchors
uv run --extra docs python scripts/check_manual.py docs/_book/smacc-manual.pdf #
PDF completeness
```

`check_links.py` fails on any dead cross-page link or `#anchor` (heading slugs differ between renderers — the main migration footgun). `check_manual.py` fails if the manual is short or missing content from its later sections, guarding against a renderer silently dropping pages (the bug behind #250).

23.6 Versioning

SMACC follows [semantic versioning](#).

- **Pre-1.0, the app is not stable.** Settings files, marker codes, the UI, and behavior may change between releases, and there are no compatibility shims. Pin one version for the duration of a study.
- **Versions are `0.x.y` until `1.0.0`.** A new `0.x.0` collects features and `0.x.y` is a smaller follow-up; no pre-1.0 bump is a stability promise.
- **`0.1.0` is the first published release** — the first with installers attached, a [release-notes](#) entry, and a working in-app update check. Earlier `0.0.x` tags predate it and are not in the release notes.
- **`1.0.0` is the first stable release.** From then on semantic versioning is binding (backward-compatible changes bump the minor/patch, breaking changes bump the major).

There are three kinds of build:

- **Stable** — `vX.Y.Z` (no suffix): a full GitHub release, badged *Latest*. The homepage download button, `/releases/latest`, and the in-app update check all resolve to it. `__version__` equals `X.Y.Z`.
- **Pre-release** — `vX.Y.Z-rc.N` (also `-alpha.N/-beta.N`): a tagged candidate for the upcoming `X.Y.Z`. Any tag with a hyphen is marked *Pre-release* on GitHub, so `/releases/latest` and the update check skip it. Allowed at any point, including the `0.x` line. Set `__version__` to the suffixed string and tag to match.
- **Development build** — the **dev tag**: a single fixed, *moving* tag (not a version number), rebuilt and republished on every code merge to `main` and marked *Pre-release*. It carries the portable `SMACC-dev.zip`

(see [Installation](#)) and is identified at runtime by the running version plus the commit it was built from, e.g. `v0.1.2 (dev build a1b2c3d)`. The word **dev** is reserved for this channel and the matching **dev** docs site; version pre-releases use `-rc/-alpha/-beta`, never a literal `-dev` suffix.

Cutting a release: bump `__version__` in `src/smacc/__init__.py` and push a matching `vX.Y.Z` tag (use `vX.Y.Z-rc.N` for a candidate). CI checks the tag equals `__version__`, builds and smoke-tests the installer, attaches it to the GitHub Release, and publishes the docs. Right after a stable release, bump `__version__` on `main` to the next target `X.Y.(Z+1)`, so development builds report the version they're heading toward rather than the one just shipped.

23.7 Project notes

- `src/` layout: the package lives in `src/smacc/`.
- The single source of truth for the version is `__version__` in `src/smacc/__init__.py`; `config.py` and the packaging metadata both read from it.
- The build/runtime Python is pinned to **3.13** in `.python-version`. The minimum OS is **Windows 10**, set by Qt 6 (PyQt6) — Qt 5 was the last line that still ran on Windows 8.1. Keep the pin unless you intend to move the Python floor.
- SMACC is distributed only as a frozen `SMACC.exe` (no PyPI), so CI tests what ships rather than a version range: the `test` job in `ci.yml` runs on the same `windows-2022` + Python 3.13 + locked dependencies as the release build (`release.yml`), across `windows-2022` and `windows-latest` rather than a Python-version matrix.