RACETRAK IMTUF

RaceTrak Operations Manual

Offline-first race tracking for ultra-marathons in country with no signal

Revision	2026-08-17 · `f49bb07`
Applies to	Field app (Android & browser), command console, portal
Audience	Part 1–2 race director and installer · Part 3 aid station and radio operators

PART 1

Installation

This part is for whoever stands the system up: one race director, once, at a desk with a network. Nobody working an aid station needs to read it.

Work through it in order. Each step is verifiable, and the verification is given, because the failure mode this system must never have is one that stays hidden until a tablet is at mile 45 with no signal.

1.1 What the system is made of

Five things run, and they are all the same event log seen from different places.

Piece	Where it runs	What it is for
Supabase Postgres	Hosted	The source of truth. Every event ends up here.
PowerSync	Hosted	Replicates Postgres to each device, and decides what each device is allowed to receive.
Field app	Android tablets	What an aid station uses. Works offline for hours.
Field app (browser)	Any phone or laptop	The same screens, for a station with signal and no tablet.
Command console	Browser, at basecamp	The whole race at once. Privileged views, roster import.
Portal	Browser, at a desk	Setting a race up: accounts, codes, imports.

Two facts about this shape are worth holding on to, because most of the procedures below only make sense in their light.

Every client holds a real database. Not a cache — a local SQLite database with the race in it. A tablet that loses signal keeps working, keeps recording, and sends its work when it can. This is why the install order below puts the backend first: a client cannot be meaningfully tested until there is something for it to replicate.

What a device may see is decided by the server, not the app. A station tablet does not hide a runner's phone number; it never receives it. That boundary lives in `powersync/sync-rules.yaml`, and §2.7 covers how to verify it rather than assume it.

> **NOTE**
>
> The event log is append-only, enforced by a database trigger. Nothing in this manual will ever ask you to edit or delete an event, because the database will refuse. Mistakes are corrected by recording a correction — see §3.6.

1.2 Before you start

You need, in this order:

1. **Node.js and npm**, on the machine doing the install. The repo is an npm workspace; everything below is an npm script.
2. **A Supabase account** and a project for the environment you are building. Free tier is sufficient for a race.
3. **A PowerSync account** and an instance pointed at that Supabase project. Free tier is sufficient.
4. **A Cloudflare account**, for hosting the three web apps.
5. **Android tablets**, if you are using them, with developer mode enabled so builds can be installed over USB or wireless debugging.

> **CAUTION**
>
> PowerSync sits **on top of** Supabase and the wiring between them is the one part general Supabase familiarity does not cover: logical replication has to be enabled on Postgres, PowerSync pointed at it, and Supabase Auth tokens threaded through so the sync rules know who is calling. Budget an unhurried block for this, well before the race — not the week of it.

1.3 Getting the code

```
git clone <your fork of the repository>
cd RaceTrak-IMTUF
npm run bootstrap
```

`npm run bootstrap` installs dependencies for every workspace at once. Do not run `npm install` inside an individual app; the workspaces share a single lockfile.

Confirm the domain logic is sound before going further:

```
npm run test
npm run lint
```

The test suite covers the event-log folds — the code that answers “where is bib 147” — and runs in seconds with no backend. If it fails, stop; nothing downstream will behave.

1.4 The backend

`docs/backend-setup.md` is the authority on this and goes deeper than a manual should. The shape of it:

1. **Create the Supabase project** and record its reference id and database password in `infra/env/<env>.env`. Start from `infra/env/example.env`; the real files are gitignored and never committed.
2. **Apply the schema.** `npm run supabase:migrate dev` runs every migration in `supabase/migrations/` in order. This creates the event table, the row-level security policies, and the append-only triggers.
3. **Deploy the pairing function.** `npm run supabase:functions dev` publishes `redeem-pairing-code`, which is how every client — tablet, console and portal — turns a code into a session. Nothing can pair until this exists.
4. **Enable logical replication** on the Postgres instance and create the read-only replication role PowerSync will connect as.
5. **Point PowerSync at it**, then deploy the sync rules with `npm run powersync:deploy dev`.

Current schema state:

Generated from the source of record. Do not edit by hand.

26 migrations, through `0026_relay_declined_targets.sql`.

> **CAUTION**
>
> `npm run powersync:deploy <env>` publishes the RBAC boundary. Never deploy it from a dirty working tree — what you deploy should be exactly what is committed, because this is the file that decides whether a phone number reaches a volunteer's tablet.

1.4.1 Verifying the backend

Do not take the console's word for it. Two checks, both cheap:

- **Replication is live.** The PowerSync instance should report a connected replication slot with lag at or near zero.
- **The append-only rule holds.** Attempt an `UPDATE` and a `DELETE` against `events` as a privileged user. Both must be refused by trigger. If either succeeds, the migrations did not all apply.

1.5 The web clients

Three separate apps, three separate Cloudflare Pages projects per environment. They are separate on purpose: the portal writes directly to Postgres and is used at a desk, while the console replicates through PowerSync and keeps working when the uplink does not.

```
npm run build:web dev          # command console
npm run build:portal dev       # portal
npm run build:fieldweb dev     # field client in a browser

npm run deploy:web dev
npm run deploy:portal dev
npm run deploy:fieldweb dev
```

Every deploy targets the project's production branch, so each app keeps one stable hostname that does not change when you deploy. Cloudflare also mints a per-deployment URL with a hash in front of it — those are build artifacts, not addresses.

> **STOP**
>
> A deployment-specific URL (`abc12345.<project>.pages.dev`) is a **different origin** from the project URL, and a client's local database belongs to the origin that created it. An operator who pairs against a hashed URL and later opens the plain one finds an empty, unpaired app, with their unsynced work stranded at the old address. Only ever hand out the plain project URL.

> **CAUTION**
>
> These apps must be served over HTTPS. PowerSync guards its database with the Web Locks API, which browsers refuse outside a secure context — over plain HTTP the app loads and then never opens its database. This is also why, in local development, you use `http://localhost:5173` and not the LAN address the dev server also prints.

1.6 The tablets

The field app is React Native under Expo. The `android/` and `ios/` directories are generated by `expo prebuild` and are not committed — never edit them by hand, because the next prebuild discards the change. Anything that must reach the native project goes through `app.json` or a config plugin.

```
RACETRAK_BUILD_VARIANT=release npm run build:android dev
adb install -r build/android/dev/app-release.apk
```

> **CAUTION**
>
> Build **release**, not debug. A debug build fetches its JavaScript bundle from a development server on the network, which means it needs a laptop within reach to start at all. That is fine on a bench and useless at an aid station. Only a release build is a real test.

1.6.1 Signing

A release build is signed with the debug keystore unless you configure a real one, which is fine for a tablet you install onto directly and unacceptable for anything distributed. Set `RACETRAK_KEYSTORE_PATH`, `RACETRAK_KEYSTORE_PASSWORD`, `RACETRAK_KEY_ALIAS` and `RACETRAK_KEY_PASSWORD` in `infra/env/<env>.env`; a production build requires all four and refuses to run without them.

```
keytool -genkeypair -keystore racetrak-release.jks -alias racetrak \
  -keyalg RSA -keysize 2048 -validity 10000
```

> **STOP**
>
> The signing key is the app's identity permanently. An app distributed under one key can **never** be updated by a build signed with another — there is no recovery, only a new app listing and every tablet reinstalled by hand. Back the keystore up somewhere a second person can reach, and keep the passwords with it.

> **NOTE**
>
> `keytool` writes PKCS12 now, and there the key password must equal the store password. Setting them differently fails at signing time with "Get Key failed: Given final block not properly padded", which does not sound like a password problem.

1.7 Verifying the whole chain

The install is not finished when things are deployed. It is finished when one tablet has carried a record all the way to Postgres. Do this once, deliberately:

1. Install a release build on a tablet that has never been paired.
2. In **Settings → Race pairing**, enter a station code from the portal.
3. Confirm the tablet syncs down: the roster appears and the station name is shown in the header.
4. Record one arrival.
5. Query the `events` table in Postgres and find that row, carrying the tablet's own device id.

That path — code, function, session, sync rules, local database, screen, upload queue, Postgres — is the system. If it completes, the install is good.

1.8 Commands, in full

Every script takes an environment (`dev`, `staging`, `prod`) and prompts before touching production.

Generated from the source of record. Do not edit by hand.

Command	Runs
`npm run bootstrap`	`./scripts/common/bootstrap.sh`
`npm run version:bump`	`node scripts/common/bump-version.mjs`
`npm run supabase:start`	`./scripts/supabase/start-local.sh`
`npm run supabase:migrate`	`./scripts/supabase/migrate.sh`
`npm run supabase:psql`	`./scripts/supabase/psql.sh`
`npm run supabase:functions`	`./scripts/supabase/deploy-functions.sh`
`npm run powersync:deploy`	`./scripts/powersync/deploy-sync-rules.sh`
`npm run console:dev`	`./scripts/web/dev-web.sh`
`npm run build:web`	`./scripts/web/build-web.sh`
`npm run studio`	`./scripts/android/studio.sh`
`npm run build:android`	`./scripts/android/build-android.sh`
`npm run build:ios`	`./scripts/ios/build-eas.sh`
`npm run deploy:web`	`./scripts/web/deploy-web.sh`
`npm run deploy:portal`	`./scripts/web/deploy-portal.sh`
`npm run submit:android`	`./scripts/android/submit-android.sh`
`npm run submit:ios`	`./scripts/ios/submit-ios.sh`
`npm run test`	`npm run test:folds && npm run test:components`
`npm run portal:dev`	`./scripts/web/dev-portal.sh`
`npm run build:portal`	`./scripts/web/build-portal.sh`
`npm run fieldweb:dev`	`./scripts/web/dev-field-web.sh`

Command	Runs
`npm run build:fieldweb`	`./scripts/web/build-field-web.sh`
`npm run deploy:fieldweb`	`./scripts/web/deploy-field-web.sh`
`npm run lint`	`eslint .`
`npm run manual:build`	`node scripts/manual/build.mjs`
`npm run docs:prod-setup`	`node scripts/docs/build-pdf.mjs docs/prod-setup.md build/docs/racetrak-prod-setup.pdf "RaceTrak — Standing up production"`
`npm run manual:check`	`node scripts/manual/check.mjs`
`npm run deploy:manual`	`./scripts/web/deploy-manual.sh`
`npm run soak:start`	`node scripts/soak/storage-soak.mjs start`
`npm run soak:check`	`node scripts/soak/storage-soak.mjs check`
`npm run test:folds`	`npm run test --workspace @racetrak/core`
`npm run test:components`	`vitest run`
`npm run test:watch`	`vitest`

PART 2

Configuration Management

This part is for the race director and anyone helping set a race up. It covers what you configure, where each setting lives, and which settings are safety boundaries rather than preferences.

The dividing line worth learning first: **the portal is where a race is set up, the console is where it is run.** The portal writes straight to Postgres at a desk with a network. The console replicates through PowerSync and keeps working when the uplink does not. They share every importer and every calculation, so they cannot disagree about what a spreadsheet row means or where a runner is.

2.1 Environments

Three environments, separated by *project* rather than by branch: a separate Supabase project, PowerSync instance and set of Cloudflare Pages projects for each. That separation is what gives each environment a stable hostname and makes it impossible to point a production tablet at development data by accident.

Environment	For
`dev`	Everyday work. Test races, throwaway data, permissive tokens.
`staging`	Optional rehearsal against production-shaped configuration.
`prod`	The race.

Configuration lives in `infra/env/<env>.env`. Only `example.env` is committed — the real files hold credentials and are gitignored. Copy the example, fill it in, and keep the production file somewhere a second person can reach it.

> **STOP**
>
> Development environments are deliberately permissive: temporary sync tokens are allowed and a preset pairing may be baked in so a client can connect before real credentials exist. Neither may be true in production. §2.9 is the checklist, and it is not optional.

2.2 Setting a race up

In the portal, in this order. Each step depends on the one before it.

1. **Create an account** and sign in.
2. **Create the race** — name and date.
3. **Add the aid stations**, in course order. Order matters: it is what makes “this runner never left the previous checkpoint” a question the app can ask.
4. **Import the roster.**
5. **Add the crews** — who is working which station.
6. **Generate the station codes** and distribute them.

All three imports — aid stations, crews and roster — accept a spreadsheet and offer a downloadable template, either blank or filled with the race’s current data. The filled template is the easier path for an edit: download, change, re-import.

2.2.1 Aid stations

Each station carries a name, its order on the course, and optionally a mile mark, opening hours, coordinates, and tags such as `radio`, `water`, `medical` or `drop_bags`. Cutoff times are set here too, and are what the field app counts down against.

2.2.2 The roster

Imported from CSV, with a mapping step: your file’s column headings are matched onto the fields the system needs — bib number, name, age, phone. Unpredictable headings are the norm and the mapper exists for exactly that; it can also combine two columns into one name.

> **NOTE**
>
> A station may assert that a bib **exists**; only the console may say who is wearing it. A tired volunteer typing a runner’s name at 3am is not a data source. This is enforced in the database, not just the interface: the insert policy refuses a row that carries a name, so identity cannot come in through the one door left open for unknown bibs.

2.2.3 Codes

Two kinds of code exist and they are not interchangeable.

Code	Grants	Given to
Station code	One station, one race. Records movements there.	Aid station and radio volunteers
Share code	The whole race, at a privileged role.	Race director, medical command

One code per station, carrying whether that station is `in`, `out` or `radio`. The code says *where* the device is; the operator chooses what the device is *doing* (§2.8). Codes are generated in the portal — one button produces one code per station — and can be sent by SMS or email.

> **CAUTION**
>
> Station codes are read aloud over radio and written on clipboards. Treat them as visible. They scope a device to one station and nothing more, which is what makes that acceptable — but a share code must never be handled the same way.

2.3 Roles, and what each one receives

This is the security model. It is enforced in the sync rules, which decide what each device ever receives, and again in row-level security in the database.

Generated from the source of record. Do not edit by hand.

Role	What it is	Sync buckets received
`aid_station`	Works a checkpoint. Receives the roster without phone numbers, the whole event log, and emergency contacts.	`race_core`
`radio`	Net Control duty at a checkpoint. Same data as an aid station; the tablet is set to Radio so the queue is its default screen.	`race_core`
`race_director`	Runs the race. Everything an aid station receives, plus runner contact details, the pairing codes and the membership list.	`race_core`, `race_contacts`, `race_admin`
`medical_command`	Medical oversight. The same privileged view as the director, for the same reason: they may need to reach a runner.	`race_core`, `race_contacts`, `race_admin`

The important property is negative and worth stating plainly: **a station device does not receive runners' phone numbers at all.** They are not hidden in the interface — the rows are absent from the tablet's database. There is nothing to leak, nothing to render by accident, and nothing recoverable by someone holding the device.

Emergency contacts are different, and are sent to station devices deliberately: the moment one is needed is at the checkpoint where somebody is on the ground, not at basecamp. Revealing one is a deliberate act that writes a permanent record of who looked (§3.6).

2.3.1 Verifying the boundary

Do not infer this from the configuration file. Confirm it:

1. Redeem a real station code and, with the session it returns, query the contacts table. It must come back empty.
2. With the same session, attempt to insert a runner. It must be refused.
3. On a paired tablet, confirm the local database holds zero contact rows.

If all three hold, both layers agree independently, which is the only form of this assurance worth having.

2.4 What a tablet is set to

Pairing decides the race and the station. Everything else is chosen on the device, in **Settings**, and can be changed mid-race.

Setting	What it controls
Race pairing	The code this device redeemed. Determines race, station and role.
Aid station	Which station this device is working.
Operator	Who is holding it this shift, and their radio call sign.
This device is	The default direction — see below.
Display	Whether runner names are shown, grid density, clock seconds.
Station mode	Station lock and its PIN.
Station Wi-Fi sync	Peer sync with the other tablet at this station.

2.4.1 Device role

Generated from the source of record. Do not edit by hand.

Setting	What the tablet says it does	When to use it
IN	Tap records an arrival	Set this on the tablet working arrivals.
OUT	Tap records a departure	Set this on the tablet working departures.
Radio	Relaying to Net Control	The only role that is offered the Radio screen.
Net Control	Taking calls from the whole net	Taking calls from the whole net at basecamp. The only role offered the Net Control screen.
Other	No default direction	Nothing is recorded without an explicit choice.

The device role sets what a single tap on the Record grid does. It is a default, not a restriction — every other action stays available behind a long press. An operator can override the direction for their shift, and that choice outranks the paired role.

> NOTE
>
> Only a device set to **Radio** is offered the Radio screen. A station running IN and OUT tablets should not have the call-in queue competing for space on either of them.

2.4.2 Display: names are off by default

A station screen is a public surface — propped on a table, photographed, visible to anyone in the tent. Runner names are personal data, and the bib is what the job actually needs, so names are off unless someone turns them on.

2.4.3 Station lock

Station lock keeps a tablet on the recording screens so it cannot be wandered away from mid-shift. It is unlocked with a PIN, or with the race admin code.

Unlocking with the **admin code** is recorded — it is a race-wide override, read over the radio, that lets somebody who is not the station operator take control of a device holding a shift's records. The station's own PIN is not recorded; that is the operator doing their own job. Overrides appear in the console under **Access**.

> **CAUTION**
>
> Station lock is an operational guard, not a security control. The PIN is stored in the clear on the device, and anyone holding the tablet can clear its data and get back in — which also destroys any records that have not yet synced. Real lockdown needs Android screen pinning or a device policy.

2.5 Two tablets at one station

The IN and OUT operators are routinely a hundred feet or more apart — too far to share a tablet, and exactly the pair who most need each other's data. When both are paired to the same station, they find each other over the station's Wi-Fi and exchange events directly, with no internet involved.

The receiving device decides what it will accept: events for another race, unknown types, or events about nobody are refused regardless of what the sender claims.

> **CAUTION**
>
> This works between **tablets only**. The browser field client has no peer sync, because a browser cannot open the necessary network connections. A station running the web client at a dark checkpoint is isolated until signal returns.

2.6 Closing a race out

Closing is a record, not a switch. It is written to the event log like everything else, carrying who closed it and the moment they did, and every client works out the race is over by reading that — so the console and every tablet reach the same answer as soon as they have the same events. A station offline at the time finds out on its next sync, which is the truth of the situation rather than a fault.

Nothing is refused because of it. Refusing late uploads would jam the queue of whichever station had been out of signal longest and lose its shift — the data that is hardest to reconstruct. So records keep arriving, and each is judged by **its own timestamp**: made before the close, it is race data however late it lands; made after, it is kept and marked post-race.

Reopening appends a retraction rather than deleting the closure, so the log shows the whole sequence. That is what makes an early press cheap.

2.7 Before a real race

Development settings are permissive by design. Every one of these must be changed before production carries real runners.

Gate	Why
A production Supabase project and PowerSync instance exist	Development data and production data must never share a database.
Temporary sync tokens **disabled** on the production instance	They let a client connect without proving who it is.
Preset pairing empty	Otherwise a build ships already paired to something.
Email confirmations **on**, with SMTP configured	Otherwise a portal signup is trusted without a verified address.
Release builds signed with a real keystore	The debug keystore is not an identity.
The build number raised before every upload	`npm run version:bump`. Play refuses a `versionCode` it has already seen — including from deleted releases and closed tracks — and a number is burned the moment Play sees it.
Offline token lifetime checked against the real race window	A 30-hour race with dark stations must not outlive its credentials.

There is a step-by-step of all of this in `docs/prod-setup.md`, with a printable PDF from `npm run docs:prod-setup`. It carries the traps that cost time the first time through — several of which fail *silently*, which is the reason it exists as a checklist rather than as prose.

> **STOP**
>
> On the last point: recording must **never** block on an expired credential. A client with no valid token is required to keep recording locally and sync later. Whatever you change about token lifetime, that behaviour has to survive, because the alternative is a station that silently stops working in the middle of the night.

PART 3

Operations

This part is for the people working a race: aid station operators, radio operators, and the director at basecamp. It assumes nothing about the rest of this manual.

Three things are true of this app, and they explain most of how it behaves.

It works with no signal. Everything you record is saved on the tablet the instant you record it. Sync happens in the background whenever there is connectivity. You never wait for the network, and losing it costs you nothing.

Nothing is ever deleted. Mistakes are fixed by recording a correction, so the log always shows both what was written and that it was put right. You cannot destroy a record by getting something wrong.

It will not stop you. Unknown bibs, duplicates, runners who appear out of order — all of these are normal at an aid station. The app tells you what it noticed and lets you decide. It never blocks a record.

3.1 Starting your shift

1. **Check the tablet is paired.** The station name appears at the top of the screen. If it says the device needs pairing, get the station code from the race director and enter it in **Settings → Race pairing**.
2. **Set your name.** Tap **Set operator** at the top, or go to **Settings → Operator**. This is how a record is attributed to you; it is not a login.
3. **Check the sync status.** Also at the top: `Synced`, `Syncing…` or `Offline`. Offline is not a problem — it is the expected state at most stations. It means what you record is waiting on the tablet, which is safe.
4. **Check what the tablet is set to do. Settings → This device is** decides what a single tap records. Make sure it matches the job you are doing.

> **NOTE**
>
> There is no password and no login. The tablet is what is authorised; your name is for attribution. A login prompt at 3am in a canyon with no signal would be a way to lose data, not a way to protect it.

3.2 The screens

Generated from the source of record. Do not edit by hand.

Tab	What it is for
Keypad	Type a bib, confirm the name, record IN or OUT. The fallback when a list is too long to scan.
Record	One square per bib. One tap logs the default direction; a long press opens everything else. The workhorse.
Log	One row per bib — in, out, called in, state. Tap a row for the full history and every correction.
Radio	The queue of movements not yet passed to Net Control, with multi-select and batch call-in. Only on a Radio device.
Net Control	Copying calls off the radio: who called, the bibs and times they read out, a read-back to correct, then acknowledge. Only on a Net Control device.
Race	Read-only reference at the station: race details, bib ranges, roster, stations, crews, cutoffs.
Settings	Pairing, operator name, device role, station lock, backend. What the device *is* — get one wrong and the record is wrong.
Preferences	Theme, contrast, button size, station colours, runner names and the time zone. Not a tab — the gear in the banner opens it, from any screen. Nothing here changes what is recorded.

Whatever you type on the Keypad survives switching tabs and coming back. An operator moving between IN and OUT halfway through a bib number does not lose the digits they had already entered.

3.2.1 Making the tablet readable

The **gear** beside the operator’s name, in the bar across the top, opens your preferences. It is on every screen, and it stays there when the device is locked to station mode — everything the lock hides can change what gets recorded, and nothing behind the gear can.

Setting	What it is for
Theme	Automatic follows the tablet. Light is tuned for direct sun; dark is for the tent after dark, when a white screen is the brightest thing in it.
Stronger contrast	Darkens borders and secondary text. For glare, and for tired eyes.
Text size	Small, Regular, Large, Larger — every screen. Separate from button size on purpose: one is whether a gloved thumb lands where it aimed, the other whether the eye behind it can read the result. This is the one that may go *smaller*, to fit more on screen.
Button size	Raises the minimum size of every tap target. It will not go below the gloved-hand minimum — to fit *more* on screen, use the magnifier on Record instead.
Square keys	On by default. Keypad keys are drawn square rather than stretched to fill, so a number key is the same shape on every screen. Turn it off to let them stretch — squares cannot use the full width of a wide tablet.
Record button size	How large **Record IN**, **Record OUT** and **Other** are drawn. Separate again: shrinking the keypad with its magnifier frees space, and this is how you spend it.
Station colors	A color per station, drawn as a stripe down the edge of the top bar, so you know which station's screen you are on before you have read anything.
Runner names, clock seconds, UTC	What the screens show.

All of it applies to **this tablet only**. It changes nothing that is recorded and nothing anyone else sees, so two people can hand the same tablet back and forth in opposite themes and the log is identical.

> **NOTE**
>
> The time zone used to be a one-tap button in the top bar. It moved here: it is a choice you make once a race, and a mis-tap switched every time on every screen between local and Zulu. The zone is still shown, on the clock.

3.3 Recording arrivals and departures

Two ways, and they are for different moments.

3.3.1 The Record grid — the fast way

A square for every bib. One tap records the direction the tablet is set to. That is the whole interaction, and it is what you will use for hundreds of runners.

- **Tap** a square to record the default direction.
- **Long press** a square for everything else — corrections, notes, DNF, the runner's history.
- **Here / Expected / All** narrows the grid. *Here* is who is standing at your station. *Expected* is everyone who could still walk in — standing here, or not yet seen — and leaves out anyone already through or out of the race.
- **Bib / Expected next** orders it. Bib order is the default, because a square that stays where your hand expects it beats a clever arrangement. *Expected next* orders by when each runner left the station before yours, which is the closest thing to an arrival order a station can have. It only appears where there is a station upstream to order by.
- **Tap = IN / Tap = OUT** at the top shows what a tap will do, and can be changed for your shift.
- The **magnifier** (🔍 − 🔍 +) sets how many squares fit across, from three to ten. Four is the default and the last size a gloved hand can reliably hit; tighter is for finding a bib in a long roster rather than tapping it first time.

> **NOTE**
>
> A tap on a runner who has **DNF'd or DNS'd does nothing**. They are out of the race, and a passage recorded against them is a timing record for somebody sitting in a tent. Long press still works, which is how a DNF typed against the wrong bib gets cleared.

Type a bib at the bottom opens a small keypad without leaving the grid. It is there for the runner your scope left out — if the grid is showing *Expected* and somebody walks in who is not on it, you do not have to go anywhere to record them. It closes again once you have, so the square that just changed color is the thing you see.

3.3.2 The Keypad — when the grid is too long

Type a bib. The runner's name appears as confirmation. Then **Record IN**, **Record OUT**, or **Other** for anything else.

Recording OUT for a runner who was never recorded IN will record both, so a runner passing straight through is one action rather than two.

The **magnifier** here sets how tall the keys are. Shrinking them is not about the keys — every point the pad gives up moves **Record IN** and **Record OUT** up towards the number you just typed and your thumb. How large those three are is set under **Record button size** in your preferences.

> **NOTE**
>
> After every record, a short **UNDO** appears at the bottom of the screen. Tapping it retracts what you just did. It is there for a few seconds and is the fastest way to fix a slip — no menus.

3.4 What the app tells you as you type

As you enter a bib, the app assesses it and tells you what it noticed. None of these stop you.

What you see	What it means	What to do
The runner's name	The bib is on the roster.	Record normally.
Not on the roster	No runner with this bib. A bandit, a typo, or a late registration.	Record it anyway. It is saved against the bib number and matched to a person later.
Already recorded IN here	This runner is already logged in at your station and has not left.	Usually a double-tap. If they really did arrive twice, record it.
Already recorded IN and OUT here	They have been all the way through your station already.	Not an error — runners do come back. But check the number first: this is what a mistyped bib looks like when it lands on somebody who has been and gone.
OUT OF THE RACE / DID NOT START	This bib belongs to somebody who has withdrawn. Shown in place of the name, in red, with where and when.	**Check the number.** Almost always a mistyped bib — the runner it names is in a tent. If it really is them, record it: you can see them and the app cannot.
Out of sequence	They never left the previous station.	Almost always a missed record upstream, not your problem to solve. Record and carry on.

> **NOTE**
>
> "Not Arrived" means the app has not seen this runner reach your station. It does not mean anything is wrong.

3.5 Radio: calling in to Net Control

The Radio screen shows movements that have **not yet been passed to Net Control**. It is a queue: work from the top.

1. Read the queue. Oldest first, except that drops jump the queue — a DNF is the thing Net Control most needs to hear.
2. Select the bibs you are about to transmit. Several at once is normal — real traffic is "bibs 47, 112 and 203 through Aid 4", not one at a time.
3. Call them in.
4. Mark them called in. They leave the queue.

Each bib appears as one line per occasion. If the same bib is recorded several times within a few minutes, those collapse into one line with a count — that is two operators confirming the same runner, not two passes.

3.5.1 When a bib comes back

A bib that genuinely appears a second time is flagged, because it is more often a correction than a second lap:

- **Amber** — your station has two times for this runner and has transmitted neither. Check which one is right before calling either in.
- **Red** — the earlier time has already gone to Net Control. The flag names the old time and when it was sent, so you can correct the record over the air rather than adding a second one.

> **CAUTION**
>
> Two operators marking the same batch as called in is not a problem and needs no coordination. It is recorded as two separate acts and the queue treats the batch as sent. Do not hold off calling something in because you think someone else might have.

3.6 Net Control: taking calls off the radio

The other end of that conversation. A station with no signal has one path off the mountain and it is the radio, so for hours at a time **what you type here is the record** — there is no other copy of it anywhere.

The screen follows radio procedure, because radio procedure exists for a reason: copy, read back, correct, acknowledge. A number misheard and written down is caught by reading it back, not by the person who said it.

1. **Log a call.** Nothing is being recorded yet.
2. **Say who is calling.** An aid station, a rover, one individual operator, or anyone else on the net. If it is a station, say which.
3. **Copy what they read.** Bibs go in a group with the time that group came through; a station calling a long list gives them in bursts, so open another group for each new time. Type the bibs however they are read — spaces, commas, anything that is not a digit separates them.
4. **Read it back.** The screen lists every bib with its time and the runner's name, and raises anything worth querying while the station is *still on the radio*: already have this one at 21:31, never left the previous station, recorded as out of the race. Each is phrased as the question to put to them.
5. **Correct anything they challenge**, then **acknowledge**. Only now is anything written.

Every bib and every time on the read-back is editable, because correcting a number is what a read-back is *for* — "negative, that was one-one-three" is one edit, not a trip back to the copy screen to work out which group it was in. The `×` beside a line drops it, for a call where two bibs were heard as one.

Times are the ones the calling station dictated, not the time you typed them. Each is marked as transcribed, so it counts as that station's own observation echoed — not as a second, independent sighting of the runner.

Anything that is not bibs — a request, a road closure, a medical query — is logged as a message instead, and needs no station and no bibs.

3.6.1 Doing it without the mouse

You are wearing a headset and the station is not going to pause while you find a checkpoint with the pointer. The whole call can be taken from the keyboard.

The letters are printed on the buttons themselves — **A** above *Aid station*, **R** above *Rover* — the way each station chip carries its number. Nothing here is a shortcut you have to remember.

The whole sequence, in order:

Key	What it does
`Enter`	On the opening screen, logs a call
`a` `r` `i` `o`	Aid station, rover, individual, other — as printed on each button
`1` … `9`	Picks the station with that number on its chip
any other letter	Jumps to the station starting with it; press again for the next one
`Enter`	Having picked who and where, puts the cursor in the bibs field
`Tab`	Moves through the fields in order
`Enter`	From the bibs field, moves to the time; from the time, opens the next group
`Shift` + `Enter`	From a time field, finishes the list and goes to the read-back
`Enter`	On the read-back, acknowledges and records the call

So a whole call is: `a`, the station number, `Enter`, the bibs, `Enter`, the time, `Shift` + `Enter`, then `Enter`. No pointer at any stage.

On the read-back, `Tab` starts at the first **bib**, because that is where a correction goes.

Letters and numbers only act while the cursor is **not** in a text field, so typing a call sign gives you the letters you pressed. `Shift` + `Enter` is the one exception — it is what you press having just typed the last time.

To drop a whole group that came out wrong, the `x` beside it removes it after confirming which bibs go with it. It is deliberately not in the `Tab` order, so it cannot be hit while typing.

> **CAUTION**
>
> Nothing you type is recorded until you acknowledge the read-back. If you close the call before then, it is gone — including anything the station already read to you. Acknowledge, then start the next call.

3.7 Fixing mistakes

Long press a bib on the Record grid, or tap its row in the Log, or use **Other** on the Keypad. Every correction is recorded as a correction: the original stays visible in the history and is marked as superseded.

Action	Use it when
UNDO (the toast)	You have just this second mis-tapped.
Correct time	The record is right, the time is wrong.
Reset bib	The records at your station for this bib should not exist. Clears them so the bib can be recorded again.
Add a note	Something needs explaining. Changes nothing about the record.
DNF	The runner stopped. Asks where — often a station behind yours.
DNS	The runner never started.
History	Everything recorded for this bib, including corrections.

3.7.1 Correcting a time

Operators log in bursts and the tablet's clock is not the runner's clock. A time recorded a few minutes late is normal and worth correcting.

When more than one time could be the one you mean — an arrival, a departure, a radio call-in — the app asks **which time is wrong** rather than guessing. Pick the one you mean, then set the correct time.

> NOTE
>
> If a correction would be overruled by another time sitting after it, the app says so before writing anything, and offers to move the others with it. Say yes if they are all the same mistake.

3.7.2 Notes

A note is for the context that otherwise lives in somebody's head until they go off shift: the bib was upside down and might be 52, the runner said their knee had gone, the time came off a radio call rather than the tablet.

A note changes nothing. It is safe to add at any moment, including in the middle of a correction, and the director sees it in the console alongside the record it explains.

3.7.3 Emergency contact

A runner's emergency contact can be revealed on a station tablet. It is deliberately a two-step action, and **it permanently records who looked and when**. That record cannot be removed by anyone.

If nobody has been set as the operator on the tablet, it asks for your name before revealing anything. The record has to say who looked, and it is kept for the rest of your shift so you are not asked again.

Use it when someone is on the ground and you need to reach their people. Do not use it out of curiosity.

> **NOTE**
>
> The runner's own phone number is not on your tablet at all, and no amount of tapping will produce it. Only the director and medical command receive it.

3.8 Working with no signal

This is the normal state at most stations, and nothing about it needs managing.

- Everything you record is saved on the tablet immediately.
- The status at the top will read `Offline`. Records accumulate safely.
- When signal returns, they upload on their own. You do not have to do anything.
- If a second tablet is working your station on the same Wi-Fi, the two share records with each other even with no internet at all.

> **STOP**
>
> Do not clear the app's data, and do not uninstall the app, while records are waiting to sync. That is the one action that destroys a station's work, and nothing can recover it. If a tablet is misbehaving, hand it to the race director rather than resetting it.

3.9 When something looks wrong

What you see	What it usually is	What to do
Screen says the device needs pairing	Not paired, or app data was cleared.	Enter the station code from the director.
Status stuck on `Offline`	No signal. Normal.	Nothing. Keep recording.
Status stuck on `Syncing…` with signal	Weak connection.	Nothing. It retries on its own.
A time shows a weekday, or a small mark after it	The record is from a **previous day**, not today.	Nothing — it is telling you that deliberately.
A runner is at a station they cannot have reached	Usually a mistyped bib somewhere upstream.	Add a note. Do not delete anything.
A bib you recorded is not in the Radio queue	Somebody already called it in.	Check the bib's history.
The tablet will not leave the recording screen	Station lock is on.	Unlock PIN, or the race admin code.

> **CAUTION**
>
> A race that runs more than 24 hours passes the same clock time twice, so a bare `21:48` is ambiguous. Every time in this app that is not from today is marked — either with its weekday, or with a small mark where a weekday will not fit. If you are reading a time to Net Control, read the day too.

3.10 For the race director on race day

The command console shows the whole race at once, and keeps working from its own local copy if basecamp loses connectivity.

- **Whole race** — every runner's position across every station, sortable, with full history per runner. Tallies, cutoffs and unknown bibs. Every count is clickable through to the bibs behind it, and the field can be filtered to one station.
- **Aid stations** — the course, and the import wizard for changing it.

- **Crews** — station personnel.
- **Roster** — CSV import with the column mapper.

3.10.1 Who is behind a cutoff

When anyone is, a panel appears above the tallies. It is silent otherwise.

It says two different kinds of thing and keeps them apart, because acting on them differs:

- **Past a cutoff with no arrival recorded** is a *fact*. The time has passed and nobody logged them through. Nothing is predicted.
- **Behind pace for the next cutoff** is an *estimate*, made from how fast that runner has covered the course so far. Somebody who has slowed to a walk, or stopped to help, will not match it. Each line shows the pace and the margin it was worked out from, because you may be deciding whether to send a person up a trail.

Runners with too little recorded to estimate are counted separately and said so. That is not the same as being on time — it means the app cannot speak to them.

3.10.2 Runners known only from the radio

A row marked **radio only** has been reported by Net Control and by no station tablet. Both causes are ordinary: that station's tablet has not synced, or there is no tablet there and the radio *is* the record. Neither is the same as a station having reported it, which is why it is marked rather than blended in.

It is the thing to look at when a station has gone quiet.

3.10.3 Closing the race out

When the race is over, **Close out the race** on the whole-race tab. Every field client stops taking entries and shows RACE COMPLETE, naming who closed it and when.

It does not cut anyone off. A station that has been offline for hours still uploads everything it recorded, and records made *before* the closing moment still count — the most remote station on the course loses nothing by syncing late. Anything recorded *after* that moment is kept too, and shown separately as post-race rather than folded into the results — **listed**, not just counted: bib, station, what was recorded, how long after the close, and how long it then took to reach the server.

Everything on that list was *recorded* after you called the race. A station uploading a backlog from during the race does not appear on it, because those are judged by their own times and counted as race data however late they arrive. So the list is a narrow question: who was still being written down after we called it. Usually a sweep tidying a station. Occasionally somebody still out there.

> **NOTE**
>
> Pressed too early? **Reopen the race** puts every station straight back to work. Nothing is deleted: the log keeps the close, the reopen, and who did each, and you can close it again when the last runner is really in.

Two jobs only the console can do:

Naming unknown bibs. A bandit recorded at a station is saved against the bib number with no name. Naming them is a console job, because a station is not permitted to assign identity. Reconciliation takes a name in the same step, and anything already on the roster without one appears in a “no name” list on the whole-race tab.

Reading the contact details. The runner’s own phone number reaches the director and medical command and nowhere else.

> **NOTE**
>
> The console gates what it shows on what the code you redeemed actually granted. If you signed in with a station code you will not see the privileged views — that is the boundary working, not a fault.

PART 4

The Shakedown

A shakedown is a rehearsal that is allowed to go wrong. Get four or five people round a table with the devices they will actually use, work through this in order, and by the end everyone has done every job at least once and the system has been exercised end to end on real hardware against the real backend.

Budget **90 minutes** for the whole thing, or an hour if you skip the last two rounds. Do it at least twice before a race: once early enough that anything broken can be fixed, and once close enough that people remember it.

The exercises are also a test. Each one says what should happen, so a facilitator can tell the difference between somebody using the app wrong and the app being wrong. Both are worth finding, and only one of them is a bug.

> NOTE
>
> Nothing here can hurt anything. It runs against the **demo race** on the development backend, which exists to be messed up. No real runner, no real timing record, and nothing anyone does today reaches a race.

4.1 What you need

People	4 minimum, 6 is comfortable
Devices	One per person. Tablets for stations, a laptop for Net Control and the director
Network	Wi-Fi everyone can join, and the ability to turn it off on at least one device
Codes	Below — print them or write them on a whiteboard
Paper	A few sheets. Radio traffic gets written down before it gets typed

4.1.1 Codes for the demo race

Code	Puts you at
`DEMOCP1`	Burgdorf
`DEMOCP2`	Duck Lake
`DEMOCP3`	Chinook
`DEMOCP4`	Snowslide
`DEMOCP5`	Willow Basket
`DEMORD`	The command console, as race director

The demo race has five stations in that order and 120 runners. Bibs 1 to 100 are the field, eighteen hours into the race. Bibs 101 to 120 are a training block that never has events, which is what the first rounds record against.

It is deliberately not empty — a roster and five stations demonstrate nothing, so it carries about a thousand events: most runners through the early stations, a thinning field, four withdrawals, a bandit and a radio queue nobody has cleared.

Its admin code is `141414`. Keep it back until round 6 — it is the override, and handing it out at the start spoils the exercise.

> **CAUTION**
>
> **Reset the demo before you start.** Every bib named below is true of a freshly built demo and only of that — a session that ran before yours will have moved some of them on.
>
> In the **portal**, under *Platform admin → Demo race*: pick **18 hours** and reset. It takes seconds.
>
> See §4.11. Reset *between* runs, never during one: it rebuilds rather than merges, so it discards whatever the group has recorded so far.

> **NOTE**
>
> A code that grants the **whole race and no station** — which is what Net Control and a roving operator want — now works directly. Net Control names the calling station on every call, so it needs no station of its own.
>
> This used to read as “not paired” on every field screen, and the workaround was to pair with some station’s code and change the role in Settings. That still works if a code is handy, but it is no longer necessary.

4.2 Setting up the room

Assign these before anyone touches a device. Swap them around on a second run — the point is that everybody has done every job once.

Role	Device	Pairs with	Then set
IN at Willow Basket	Tablet	`DEMOCP5`	This device is → **IN**
OUT at Willow Basket	Tablet	`DEMOCP5`	This device is → **OUT**
Radio at Willow Basket	Tablet	`DEMOCP5`	This device is → **Radio**
Net Control	Laptop	`DEMOCP1`	This device is → **Net Control**
Race director	Laptop	`DEMORD`	— (console, not the field app)

> **NOTE**
>
> Willow Basket rather than the first station, and for a reason worth knowing. Eighteen hours in, everyone has long since left Burgdorf, so nobody is standing there and the out-of-sequence warning in round 2 has nothing to fire on. Willow Basket is the last station: the runners still at Snowslide have arrived upstream and not left, which is exactly the condition that warning is about. It is also the quietest station, which suits a room full of people learning.

Everybody who is on a field device does this first:

1. Open the field client and go to **Settings → Race pairing**. Enter your code.
2. Wait for the roster to arrive. The station name appears at the top.

3. **Settings → Operator**, or tap **Set operator** at the top of any screen, and put your own name in. Every record you make today will carry it.
4. **Settings → This device is** — set the role from the table above.
5. Look at the top of the screen and say out loud what it says: `Synced`, `Syncing…` or `Offline`.

> **NOTE**
>
> Step 5 matters more than it looks. The single most useful habit an operator can have is knowing, without being asked, whether their tablet is talking to anything. Ask them again at random during the session.

4.3 Round 1 — The ordinary case

The thing that happens five hundred times and must never be interesting.

1. **IN** records bibs **101 to 110** arriving. Use the Record grid: one tap each. These have no history anywhere, so nothing should warn about anything.
2. **OUT** records the same ten leaving.
3. Both operators watch the **Present** count at the top as they go.

Expect: the count climbs to ten as IN works, and falls back to zero as OUT does. The two tablets show each other's work within a few seconds.

Why: this is the whole job. If it is not effortless here, it will not be effortless at 3am. Time it — ten runners should take well under a minute.

4.3.1 Then the keypad

1. **IN** switches to the **Keypad** tab and records bib **111** arriving by typing. Watch the display as you tap: it must read `1`, `11`, `111`. A bib with a repeated digit is the one worth watching, because a keypad that swallowed the second tap would look like a mis-tap rather than a fault.
2. Type bib **112**, then switch to the **Log** tab and back to **Keypad**.

Expect: the digits are still there. An operator moving between screens mid-runner does not lose the number they were part-way through.

4.4 Round 2 — The four things the app tells you

None of these stop anyone. The app reports what it noticed and the operator decides. Make sure everyone sees all four.

1. **A match.** Type bib **113** on the Keypad. The runner's name appears.
2. **Not on the roster.** Type bib **9999**. It says so — record it anyway. The roster stops at 120, so this is a bandit.
3. **A duplicate.** Record bib 113 arriving, then record it arriving again.
4. **Out of sequence.** Record bib **5** arriving at Willow Basket. That runner arrived at Snowslide and never left, which is what the warning is about.

Expect: four different messages, word for word, and none of them prevents the record:

Step	What the screen says
Match	**On roster** ✓ — or the runner's name, if names are switched on
Not on roster	**Not on roster**, then *Will be saved as unknown bib 9999 — reconcile later*
Duplicate	**Already recorded IN here at** *hh:mm*
Out of sequence	**No OUT recorded from Snowslide**

> **NOTE**
>
> Step 4 only fires because bib 5 was last seen *arriving* at Snowslide. A runner with no record at the previous station at all produces no warning, and deliberately so — somebody who skipped a checkpoint entirely is a course question for the director, not something to nag an operator about mid-flow. Bibs 32, 41, 50 and 59 are in the same state if you want a second go.

Why: every one of these is normal at a real station. Bandits, late registrations, a double tap, a missed record upstream. An operator who thinks the app is broken when it says "not on roster" will stop recording, and a station that stops recording is worse than any of these.

> **NOTE**
>
> Bib 9999 stays on the list of unrecognised bibs until the director names it in round 6. Leave it there.

4.5 Round 3 — Radio and Net Control

The half of the system that exists because most stations have no signal. Put the Net Control laptop out of earshot if you can, or at the far end of the table, and make people actually speak.

1. **Radio** opens the **Radio** tab. The queue holds everything recorded in rounds 1 and 2 that has not been called in.
2. **Radio** selects four or five bibs and reads them aloud to Net Control, with times: *"Net Control, Willow Basket, bibs 12, 27, 44 out at 14:02, 14:04, 14:05."*
3. **Net Control** taps **Log a call**, picks **Aid station** → **Willow Basket**, and types each bib and time as they hear it.
4. **Net Control** taps **Read back** and reads the list back aloud.
5. **Radio** deliberately challenges one of them: *"Negative, bib 2 was 14:14."*
6. **Net Control** corrects that time in the read-back and reads it again.
7. **Net Control** taps **Acknowledge and record**.
8. **Radio** now marks those bibs called in on the tablet.

Expect: the corrected time is what gets recorded. The Radio queue empties of the bibs that were called in.

Why: this is radio procedure, and the app is shaped around it. Nothing Net Control types is recorded until it has been read back and acknowledged, because voice over a noisy net is lossy and the read-back is what catches it. The times recorded are the ones the station said, not the ones Net Control typed them at.

4.5.1 Traffic that is not bibs

1. **Radio** calls something in that has nothing to do with a runner: *"Net Control, Willow Basket, we're low on water."*
2. **Net Control** logs a call with a source and **no bibs**, using the **Anything else** box, and acknowledges it.

Expect: it records with no bib and no runner attached.

4.5.2 And from someone who is not a station

1. Somebody plays a rover: *"Net Control, Rover 2, I have bib 40 through Chinook at 15:20."*
2. **Net Control** picks **Rover**, types the call sign, enters the bib and time, and is then asked **which station** the bib passed. Pick Chinook.

Expect: you cannot reach the read-back until a station is chosen.

Why: a bib passed *somewhere*, and everything downstream is keyed on which station. The app asks rather than guessing, because a guessed checkpoint is a location in the record that nobody observed.

4.6 Round 4 — Fixing things

Everyone should break something on purpose and put it right. This is the round that decides whether people trust the app enough to keep recording when they are unsure.

1. **Undo.** Record a bib by mistake and tap **UNDO** on the toast before it disappears. It has about five seconds.
2. **Correct a time.** Long press a bib on the Record grid → **Correct time**. If the runner has both an in and an out, the app asks *which* time is wrong. Pick one, change it, save.
3. **Reset a bib.** Long press → **Reset bib**. Then record it again from scratch.
4. **Add a note.** Long press → **Add a note**: *"bib was upside down, might be 52."*
5. **Read the history.** Long press → **History**. Find the correction you made and the note.

Expect: the history shows what was originally recorded *and* that it was corrected, with your name on it. Nothing has been deleted.

Why: the log is append-only, which is what makes it safe to let a tired volunteer fix their own mistakes. Say this out loud to the group — people are noticeably more willing to correct things once they know they cannot destroy anything.

4.7 Round 5 — Losing the network

The normal state at most stations, and the one people find most alarming until they have seen it.

1. **IN** turns Wi-Fi off on their tablet.
2. Record bibs **115 to 119** arriving.
3. Watch the top of the screen: it now says `Offline`.
4. Ask the **director** whether those runners have appeared in the console. They have not.
5. Turn Wi-Fi back on and watch.

Expect: recording never hesitates while offline. Within seconds of the network returning, the five arrivals appear in the console.

Why: nothing about being offline needs managing. The most valuable thing an operator can learn today is that `Offline` is not a problem and not a reason to stop, and that they never have to press anything to make records send.

> **STOP**
>
> This is the moment to say the one thing that can actually lose data: **never clear the app's data or uninstall the app while records are waiting to send.** If a tablet is misbehaving, hand it to the director rather than resetting it.

4.7.1 If you have two tablets at one station

Put both on the same Wi-Fi with **no internet** — a phone hotspot with mobile data off works. Record on one and watch it appear on the other. Stations run IN and OUT a hundred feet apart, and they share a log directly when there is no uplink.

4.8 Round 6 — The director's jobs

Now the console, on the laptop paired with `DEMORD`.

1. **The whole race.** Find a runner and expand their history. Every station, every correction, who made it.

2. **Name the unknown bib.** Bib 9999 from round 2 is on the reconciliation list. Add it to the roster, with a name.
3. **Break glass.** On a *tablet*, long press a runner → **Emergency contact** → reveal it. If the tablet has no operator name set, it asks who is looking — that is deliberate.
4. **Find the access record.** In the console, open the **Access** tab. The reveal is there, with who did it, when, and on which device.
5. **Lock a station.** On a tablet, **Settings** → **Station mode**, set a PIN, tap **Lock this device**. Try to reach Settings — you cannot. Unlock with the PIN.
6. **Override the lock.** Lock it again, then have somebody else unlock it with the race admin code `141414` instead of the PIN.
7. **Find the override.** Back in the console's **Access** tab, the admin unlock is recorded too. The station's own PIN is not — that is the operator doing their own job.

Expect: every privileged action is in the Access tab and cannot be removed from it, including by the director.

Why: an audit trail nobody can review is not one. Show the group that revealing a contact is permanently attributed — it is the most effective way to make sure it is only used when somebody is on the ground.

4.9 Round 7 — Closing the race

Do this last. It stops everyone recording.

1. **Director**, in the console, on the whole-race tab: **Close out the race**. Type your name and confirm.
2. Everybody look at their tablet.
3. Try to record something.
4. **Director**: **Reopen the race**.
5. Everybody look again.

Expect: every field device shows **RACE COMPLETE**, naming who closed it and when, and the recording screens are gone. Reopening brings them all back without anyone re-pairing.

Why: worth doing in front of people so that when it happens for real nobody thinks their tablet has failed. Point out that reopening is one action — closing early is a recoverable mistake, not a disaster.

> **NOTE**
>
> A station that is offline when the race closes carries on recording and finds out on its next sync. Anything it recorded *before* the close still counts, however late it arrives. Nothing that was recorded is lost by closing.

4.10 What should not happen

If you see any of these, it is the app and not the operator. Write down what was on screen, who did what, and roughly when.

Signature	Why it matters
A tap on the grid records nothing, or records the wrong direction	The core interaction
The Present count disagrees with what the tablets have recorded	A fold is wrong
A correction reverts after a few seconds	A write is not landing
Records made offline never arrive after the network returns	The queue is stuck
Two tablets at one station disagree about a runner for more than a few seconds	Sync or relay
A time shows without a day when it is not from today	Reading it to Net Control would be a day out
A station tablet shows a runner's own phone number	The RBAC boundary. Stop and report it

That last one has been verified three separate ways and should be impossible. Report it immediately anyway.

4.11 Resetting for another run

The demo rebuilds in one action, so a second group starts from exactly the state the first one did — and so do the bib numbers above.

From the portal, which is the way to do it in a room: *Platform admin → Demo race*, choose how far in, reset. Platform admins only.

Or from a database prompt, if you are already at one:

```
npm run supabase:psql dev -- -c "select reset_demo_race(18);"
```

The number is how far into the race to catch it. **18 is what this script assumes** and what a shakedown wants: a radio queue to work, runners standing at stations, cutoffs in all three of their states.

0 gives the start line instead — roster, stations and codes with nothing recorded. That is the better setting for showing the app to someone who is not going to work a station, because everything on screen is then something they did themselves. It is the wrong setting for this script: four of the seven rounds need the seeded state and would have nothing to show.

> **CAUTION**
>
> It rebuilds, it does not merge. Everything anybody recorded in the demo is gone, which is the point — but it means resetting mid-session throws away the group's own work, including anything they were about to look at in the console. Reset between runs, not during one.

Resetting is also the honest fix if the bib numbers above have stopped behaving. They are true of a freshly built demo; a session that ran before yours will have moved some of them on.

4.12 Debrief

Ten minutes at the end, and worth more than the last exercise.

- What did you expect to happen that did not?
- What did you have to be told that the screen should have told you?
- What would you not want to do at 3am, cold, with gloves on?

- Which screen would you rather never see again?

Write the answers down. The most useful findings from a session like this are almost never bugs — they are places where the app is technically correct and still asks too much of somebody who has been awake for twenty hours.

RaceTrak Operations Manual · revision 2026-08-17 (`f49bb07`). Sections marked as generated are produced from the source of record and verified by `npm run manual:check`.