Chapter 13: Maintenance and the Cleanup Script

A self-hosted Ensemble installation requires very little ongoing maintenance. The database manages itself, schema updates apply automatically on upgrade, and the application generates no log files of its own. The two things that do need occasional attention are accumulated stale data from inactive users and the SQLite database file itself, which benefits from a periodic backup.

This chapter covers the cleanup script `ensemble_cleanup.php`, how to run it manually or on a schedule, how to upgrade Ensemble to a new version, and how to keep a safe copy of the database.

13.1 The Cleanup Script

`ensemble_cleanup.php` is a maintenance script that removes stale data from the database and filesystem. It is designed to be run periodically — weekly or monthly — on installations that serve multiple users over time.

The script performs three tasks:

- **Inactive user removal** — deletes users who have not been seen for longer than the configured inactivity threshold (default: 365 days). When a user is removed, the cascade deletes all their associated data: outfits, shared links, remember-me login tokens, and uploaded outfit images.
- **Expired token cleanup** — removes expired remember-me login tokens from all users, regardless of activity. These tokens accumulate over time as users log in from different browsers; cleaning them up reduces the size of the `remember_tokens` table.
- **Orphaned image cleanup** — scans the `uploads/` directory for image files that no longer have a corresponding record in the `outfits` table, and deletes them. These can accumulate if outfits are deleted after their images were uploaded.

> **For personal use:** If you are running Ensemble for yourself and perhaps a small group of friends, you may never need the cleanup script. It is most useful for shared public installations where new users register and then stop using the service.

13.2 Installing the Cleanup Script

`ensemble_cleanup.php` is included in the Ensemble release package. Deploy it alongside the other PHP files in your web root:

```
/srv/ensemble/www/ensemble_cleanup.php
```

Set the same permissions as the other PHP files:

```
chmod 644 /srv/ensemble/www/ensemble_cleanup.php
```

The script can be run in two ways: from the command line (CLI) or via HTTP. Web access is disabled by default and must be explicitly enabled; CLI access is always available.

13.3 Configuration

The cleanup script has two configuration constants at the top of the file. Open the file in a text editor to review or change them:

```
nano /srv/ensemble/www/ensemble_cleanup.php
```

INACTIVITY_DAYS

The number of days of inactivity after which a user account is eligible for removal. The default is `365` — one year. A user is considered inactive if the `last_seen` timestamp on their account (the time of their HUD's most recent heartbeat) is older than this threshold.

```
define('INACTIVITY_DAYS', 365);
```

Increase this threshold if you want to retain accounts for longer. Decrease it if you are running a high-traffic shared installation and want to reclaim space sooner. Set it to a very large number (e.g. `9999`) to effectively disable inactive user removal while still running the token and orphan cleanup.

> **Note:** Inactivity is measured from the HUD's last heartbeat, not from when the user last logged in to the web panel. A user whose HUD is not worn but who still logs in to the web panel periodically will still appear inactive to this script. If you want to protect specific accounts from removal, the safest approach is to set `INACTIVITY_DAYS` high and review the dry-run output before committing.

WEB_ALLOWED

Controls whether the cleanup script can be triggered via an HTTP request. The default is `false` — web access is disabled.

```
define('WEB_ALLOWED', false);
```

Web access is intentionally off by default because the cleanup script is destructive and has no authentication of its own — any visitor who can reach the URL could trigger it. If you want to run it via HTTP (for example from a cron job on a remote server that cannot SSH in), set this to `true` and ensure the URL is not publicly known or guessable.

> **Security note:** If you enable web access, the script URL is `https://yourdomain.example.com/ensemble_cleanup.php`. Anyone who knows this URL can run the cleanup. Consider adding a secret token query parameter check inside the script, or restricting the URL by IP address in your web server configuration, before enabling web access on a public installation.

13.4 Dry-Run Mode

Before running the cleanup for real, use dry-run mode to see exactly what would be removed without deleting anything. This is strongly recommended before the first run on an active installation.

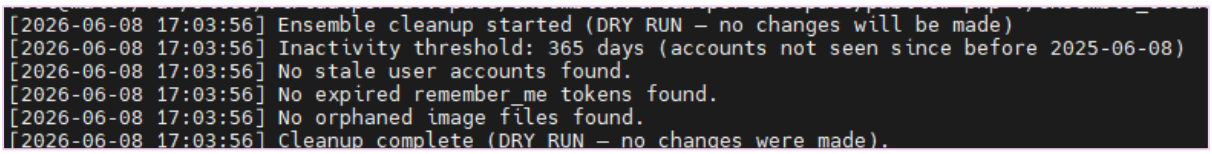
To run a dry-run from the command line:

```
php /srv/ensemble/www/ensemble_cleanup.php --dry-run
```

To run a dry-run via HTTP (requires `WEB_ALLOWED = true`):

```
https://yourdomain.example.com/ensemble_cleanup.php?dry_run=1
```

The dry-run output lists every user that would be removed, every token that would be deleted, and every orphaned image file that would be cleaned up. Review this output carefully before committing to a real run.



Terminal output of ensemble_cleanup.php --dry-run, showing a list of users that would be removed, expired tokens counted, and orphaned files identified

13.5 Running the Cleanup

Manual run from the command line

To run the cleanup and commit all changes:

```
php /srv/ensemble/www/ensemble_cleanup.php
```

The script outputs a summary of what was deleted: number of users removed, number of tokens cleaned, number of orphaned images deleted. All operations run inside a database transaction — if any step fails, the entire cleanup is rolled back.

Running via HTTP

With `WEB_ALLOWED = true`, visit:

```
https://yourdomain.example.com/ensemble_cleanup.php
```

The page outputs plain text showing the same summary as the CLI version. This is suitable for manual triggering from a browser or for use with a remote monitoring or cron service.

Cron schedule

For automated periodic cleanup, add a cron entry. To run monthly at 03:00 on the first of each month:

```
0 3 1 * * php /srv/ensemble/www/ensemble_cleanup.php >>
/var/log/ensemble_cleanup.log 2>&1
```

This pipes the output to a log file so you can review what was cleaned each time. The log file will grow over time; rotate it with `logrotate` or truncate it periodically.

To run weekly on Sunday at 03:00:

```
0 3 * * 0 php /srv/ensemble/www/ensemble_cleanup.php >>
/var/log/ensemble_cleanup.log 2>&1
```

Add the cron entry with:

```
sudo crontab -e
```

Which user to run cron as: Run the cron job as the same user that owns the `data/` and `uploads/` directories (`www-data` in the standard setup). To edit the `www-data` crontab: `sudo crontab -u www-data -e`. Alternatively, if the `data/` and `uploads/` directories are writable by your own user, add the cron entry to your own crontab.

13.6 What the Cleanup Removes in Detail

Understanding exactly what is deleted helps you decide on the right threshold and review dry-run output confidently.

Inactive users

A user is eligible for removal if their `last_seen` value in the database is older than `INACTIVITY_DAYS` days ago. When a user is removed, the following is deleted as a cascade:

- All outfit records for that user (including all properties, sequences, and removal point configurations).
- All shared links created by that user. Any visitor currently holding one of those link URLs will receive a “Link not found” page.
- All remember-me login tokens for that user.
- The user’s uploads subdirectory and all image files within it (`uploads/<user_uuid>/`).
- The user record itself from the `users` table.

The user’s account on the web panel ceases to exist. If the same avatar wears the HUD again after their account has been removed, the HUD will register a new account as if connecting for the first time, and a new temporary password will be sent.

Expired remember-me tokens

Remember-me tokens are created when a user logs in and are stored in the `remember_tokens` table. They expire after 3,650 days (ten years) — effectively permanent unless revoked by signing out. The cleanup removes tokens whose `expires_at` timestamp has passed, which in practice means tokens from users who explicitly set a shorter expiry or whose system clock had an issue at creation time. This step is mostly precautionary.

Orphaned image files

Outfit images are stored in `uploads/<user_uuid>/<filename>.jpg`. An orphaned image is a file in the uploads directory that does not correspond to any outfit record in the database. Orphans accumulate when:

- An outfit is deleted from the web panel after its image was uploaded.
- A restore replaces outfits and the old image files are left behind.

- An image upload succeeded but the associated outfit record was subsequently removed.

The orphan check only removes files, never directories. The per-user subdirectory (`uploads/<user_uuid>/`) is left in place even if all its images are removed — it will be removed as part of the user cascade if the account is later cleaned up.

13.7 Database Maintenance

Backups

The SQLite database file at `DB_PATH` is the single most important file in your installation. It contains all user accounts, outfit data, links, and settings. Back it up regularly.

Because SQLite uses WAL (Write-Ahead Log) mode, a simple file copy is safe even while the application is running. The recommended approach is a cron job that copies the file daily:

```
# Run daily at 02:00, keep 30 days of backups 0 2 * * * cp
/srv/ensemble/data/ensemble.db \
/srv/ensemble/backups/ensemble_$(date +\%Y\%m\%d).db  # Clean up
backups older than 30 days 5 2 * * * find /srv/ensemble/backups/ -
name 'ensemble_*.db' \     -mtime +30 -delete
```

Create the backups directory first:

```
sudo mkdir -p /srv/ensemble/backups sudo chown www-data:www-data
/srv/ensemble/backups
```

> **Off-site backups:** Store a copy of the database file off the server periodically — at minimum weekly. A simple rsync to another machine, or uploading to object storage (S3, Backblaze B2, etc.), provides protection against server loss. The database file is small (typically under a few megabytes even with thousands of outfits).

VACUUM

SQLite databases can accumulate unused space after deletions. Running `VACUUM` compacts the database file and reclaims this space. This is worth running after a large cleanup:

```
sqlite3 /srv/ensemble/data/ensemble.db 'VACUUM;'
```

On a small personal installation this is rarely necessary — the database stays small regardless. On a multi-user installation that regularly removes inactive users, periodic vacuuming keeps the file compact.

> **Install sqlite3:** The `sqlite3` command-line tool may not be installed by default. Install it with: `sudo apt install sqlite3`. It is not required for Ensemble to run — only for manual database operations like VACUUM.

13.8 Upgrading Ensemble

Ensemble is upgraded by replacing the PHP files with newer versions. The database schema is managed entirely by `db.php`, which applies any new columns or tables

automatically on the first request after an upgrade. No manual SQL is required and there is no migration command to run.

Upgrade steps

1. Download the new release from the OpenSimWorld Script Library.
2. Download a backup from the web panel (**Backup › Download Backup**) and also copy the database file directly as a second safeguard.
3. Copy your existing `config.php` to a safe location — it is not replaced by the upgrade files, but keeping a copy avoids accidental overwrites.
4. Replace the PHP files in your web root with the new versions. The new release will contain: `index.php`, `api.php`, `auth.php`, `backup.php`, `db.php`, `image.php`, `view.php`, `ensemble_cleanup.php`, and the `themes/` directory.
5. Put your saved `config.php` back in place. Your `DB_PATH`, `UPLOADS_DIR`, `LINKS_MAX`, and `ACCESS_CODE` settings are preserved.
6. Set file permissions on the new files:

```
chmod 644 /srv/ensemble/www/*.php
```

7. Open the web panel in a browser. The first request triggers `db_bootstrap()`, which applies any new schema changes automatically. The application is then ready to use.

There is no downtime procedure. The upgrade completes on the first PHP request, typically in milliseconds.

Community themes during upgrade

If you have installed community themes (additional subdirectories in `themes/`), these are not affected by upgrading the core files because the `themes/` directory is separate. The built-in `default/` and `dark/` theme directories will be replaced by the new release, which updates their CSS. If you have customised the built-in themes directly — rather than creating separate theme folders — back those up before upgrading.

Upgrading the HUD

The PHP files and the HUD scripts are versioned independently. Upgrading the web panel does not require a simultaneous HUD upgrade, and vice versa. The web panel and HUD maintain backward compatibility across minor versions.

When a new HUD release is available, distribute the new HUD object to users in-world. Users can replace their existing HUD — it will perform a factory reset on first wear by a new owner, or simply reconnect with updated script versions if worn by the same avatar. The web panel will display the updated script version numbers in Settings › Account once the new HUD has checked in.

13.9 Monitoring a Running Installation

Ensemble does not produce its own application logs beyond what PHP and your web server already write. The things worth checking occasionally on a running installation:

- **Web server error log** — `/var/log/nginx/error.log` or `/var/log/apache2/ensemble_error.log`. Look for recurring PHP errors or 502/503 responses that might indicate PHP-FPM issues.
- **Database file size** — run `ls -lh /srv/ensemble/data/ensemble.db`. Growth is expected over time; very rapid growth may indicate a bug in the heartbeat flow creating duplicate records.
- **Uploads directory size** — `du -sh /srv/ensemble/uploads/`. Each user's images take a few megabytes at most. Unusually large size may mean orphaned files have accumulated; the cleanup script will address this.
- **PHP-FPM or Apache process health** — `sudo systemctl status php8.2-fpm` or `sudo systemctl status apache2`. Both should always show `active (running)`.
- **Disk space** — `df -h`. The most common cause of a non-functional installation on a VPS is a full disk. The database, uploads, and web server logs are the main consumers.

For a personal or small-group installation, a monthly glance at these metrics is sufficient. For a larger shared installation, consider setting up basic server monitoring (Netdata, Uptime Kuma, or similar) to alert you to problems automatically.

This concludes Part 3. The Appendix covers creating custom themes for Ensemble.