Mitentwickeln

Seitenstand 1.03 (2026-11-24), gilt für Version 5.12

MobaLedLib Studio lebt von der Mitarbeit vieler. Du kannst Fehler beheben, Funktionen ergänzen, Übersetzungen verbessern, Anleitungen schreiben oder neue Versionen testen. Diese Seite erklärt, **wie du mithilfst**, **was du beachten musst**, wie dir eine **KI** dabei helfen kann und **wie deine Arbeit in das Programm kommt**.

Du möchtest nur einen Fehler melden? Dann lies Fehler melden. Das ist die einfachste Form der Mitarbeit.

Wie kannst du helfen?

Aufgabe	Was du brauchst
Neue Versionen testen	Nur die Betatest-Checkliste (`install/BETATEST_Checkliste.md`) und eine Lichtmaschine. Besonders gebraucht: Hardwaretests, Windows, macOS, Linux, Raspberry Pi, iPad.
Übersetzen	Die Datei `csv/Web_Languages.csv` enthält alle Texte in sieben Sprachen. Du kannst Texte prüfen und verbessern, ohne zu programmieren.
Wiki und Anleitungen	Ergänzungen und Korrekturen an den Wiki-Seiten. Schick Vorschläge als Issue.
Fehler beheben, Funktionen bauen	Python-Kenntnisse (oder eine KI, siehe unten) und ein GitHub-Konto.

Beim **Testen** und **Übersetzen** kommst du ohne Programmierung aus. Schreibe deine Ergebnisse als Issue oder in das Forum.

Der Weg einer Änderung im Überblick

1. **Absprechen:** Lege einen Issue an (oder kommentiere einen vorhandenen), was du ändern möchtest. So erfährst du früh, ob jemand anderes schon daran arbeitet.
2. **Kopie anlegen: Fork** des Repositories auf GitHub, dazu ein **Branch** für deine Änderung.
3. **Ändern und testen:** Auf Basis der **neuesten Release-Version**, nach den Regeln unten.
4. **Pull Request:** Du schickst die Änderung als **Pull Request** (PR) an das Hauptrepository.
5. **Prüfung und Übernahme:** Der Entwickler prüft die Änderung und übernimmt sie in den Entwicklungsstand (siehe unten).
6. **Test und Release:** Die Betatester prüfen sie. Mit der nächsten Version kommt sie zu allen Anwendern.

Die Technik in Kürze

- **Repository:** github.com/haroldlinke/WebMLL. Das Programm liegt im Ordner `python`. Der Zweig `main` enthält den **letzten Release**; seine Nummer steht im Tag, zum Beispiel `V3.5.0L3-3.85` (der Teil nach dem Bindestrich ist die Programmversion).

- **Sprache:** Python. Die Oberfläche läuft im Browser mit **NiceGUI** (Ordner `_web`, Hauptprogramm `webmobaledlib.py`). Die Fachlogik stammt aus dem Excel-Original (VBA), das nach Python übersetzt wurde (Ordner `proggen`, `pattgen`, `mlpyproggen`, `ExcelAPI`). Der Ordner `headless` ersetzt die Fenster der Tk-Oberfläche durch Code ohne Fenster.
- **Übersicht für Entwickler:** Die Datei `KI_prompt/KI_ENTWICKLUNG_Prompt.md` im Repository beschreibt alle Ordner, bekannte Fallstricke, die Arbeitsregeln und die Tests ausführlich. Lies sie, bevor du größere Änderungen machst. Sie ist auch das Briefing für eine KI.
- **Zusätzlich lesen:** `HOC_intern.md` (warum wurde was geändert) und `install/ANLEITUNG_Installation.md`. Beide liegen im Release-Paket.

Was du beachten musst

Vor dem Ändern

- **Basis:** Arbeite immer auf der **neuesten Release-Version** (Zweig `main`), nicht auf einem alten Stand. Schreibe in den PR, **auf welcher Version** er beruht. Der Entwickler arbeitet oft schon an einer neueren Version; er führt deine Änderung dann mit seinem Stand zusammen.
- **Klein und gezielt: Ein Thema pro Pull Request.** Kein Umformatieren ganzer Dateien, keine Umbenennungen nebenbei, keine neuen Fremdbibliotheken ohne Absprache. Kleine PRs werden schnell geprüft, große warten lange.
- **Namen prüfen:** Neue Funktionen und Konstanten dürfen keinen vorhandenen Namen überschreiben (ein Beispiel aus der Vergangenheit: eine Liste `LANGUAGE_NAMES` wurde überschrieben und hätte Sprachen zerstört).

Im Code

- **Kopf einer Python-Datei:** Jede geänderte Datei hat einen Kopf mit Name, Pfad (relativ zu `python`), **Version**, Datum, Autor, Lizenz (LGPL V2.1) und einer **History of Change**. Erhöhe die Version und ergänze eine Zeile in der History. Gibt es noch keinen Kopf, füge einen ein (Muster: eine beliebige Datei im Ordner `_web`).
- **Jede geänderte Zeile bekommt einen Kommentar mit der neuen Version**, zum Beispiel `# v1.15`. **Gelöschte Zeilen bleiben als Kommentar stehen** und bekommen ebenfalls die Version. So sieht man jede Änderung im Code, ohne Vergleichsprogramm.
- **Zeilenenden** nicht ändern: Das Projekt hat Dateien mit CRLF und mit LF. Der Editor muss das vorhandene Zeilenende behalten.
- **Texte für Anwender** sind **deutsch** und stehen in `tr("…")`. Der deutsche Text ist der Schlüssel in `csv/Web_Languages.csv`. Trage neue Texte dort ein (sieben Spalten: Deutsch, Englisch, Niederländisch, Französisch, Italienisch, Spanisch, Dänisch). Fehlt eine Sprache, gilt Englisch. Schreibe nur die Sprachen, die du sicher kannst, und sag es im PR.
- **Namen aus der Oberfläche** (Knöpfe, Reiter, Menüs) im Text immer **genau so** schreiben, wie sie im Programm heißen. Beispiel: **Lichtmaschine**, **Mit Lichtmaschine verbinden**, **WLAN (OTA)**.
- **Anwender sollen nie von Hand in wichtigen Dateien editieren müssen** (zum Beispiel `start_windows.bat`). Einstellungen gehören als Option in die Oberfläche (**Programm Einstellungen**).
- **Symbole** (Ordner `icons` und `icons_gross`) gehören nicht in einen PR. Die Dateinamen müssen exakt wie in der Makrotabelle geschrieben sein.

Testen

- **Syntax:** `python -m py_compile datei.py` für jede geänderte Datei.
- **Start:** Das Programm muss starten (`python webmobaledlib.py` oder das Startskript für dein System) und die Seite, die du geändert hast, muss sich öffnen lassen.
- **Dein Bereich:** Probiere die Funktion selbst aus, auch Randfälle (leere Eingaben, Sonderzeichen, Sprache auf Englisch stellen).
- **Schreibe ehrlich, was du nicht geprüft hast** (zum Beispiel „nur unter Windows, nicht mit Hardware“). Das ist wichtig und kein Makel: Der Entwickler und die Betatester prüfen das gezielt nach.
- **Dokumentation:** Wenn deine Änderung sichtbar ist, schreibe in den PR zwei, drei Sätze für die Release-Hinweise (**Neue Funktionen** oder **Korrekturen**) und, wenn du kannst, einen Punkt für die Betatest-Checkliste (was soll geprüft werden?).

Pull Request anlegen

1. Melde dich bei GitHub an und öffne github.com/haroldlinke/WebMLL. Klicke oben rechts auf **Fork**.
2. Hole deine Kopie auf den Computer: `git clone https://github.com/<dein-Name>/WebMLL`. (Oder bearbeite einzelne Dateien direkt im Browser mit dem Stiftsymbol; GitHub legt dann Fork und Branch für dich an.)
3. Lege einen **Branch** an: `git switch -c mein-thema`. Der Name sagt, worum es geht, zum Beispiel `fix-zeitangabe-komma`.
4. Ändere die Dateien nach den Regeln oben und teste.
5. `git add` und `git commit` mit einer **kurzen, klaren Beschreibung** (was und warum). Dann `git push -u origin mein-thema`.
6. Auf GitHub erscheint **Compare & pull request**. Klicke darauf, prüfe, dass der **Zielzweig** `main` des Repositories `haroldlinke/WebMLL` ist, und fülle die Beschreibung aus (Vorlage unten).
7. Klicke auf **Create pull request**. Du kannst den PR danach jederzeit ergänzen (neue Commits im selben Branch) oder Fragen im PR beantworten.

Vorlage für die Beschreibung des Pull Requests:

```
**Basis:** Programmversion, auf der die Änderung beruht (zum Beispiel 3.85)
**Zugehöriger Issue:** #123 (falls vorhanden)

**Was ändert der PR und warum?**
(zwei bis fünf Sätze)

**Geänderte Dateien:**
- _web/web_xxx.py (Version 1.15): ...
- csv/Web_Languages.csv: 3 neue Texte (Deutsch, Englisch)

**Getestet:**
- Windows 11, Chrome, Python 3.12: ...
**Nicht geprüft:**
- Hardware, macOS, Raspberry Pi

**Text für die Release-Hinweise:**
- Korrekturen: ...
```

Wie deine Änderung zusammengeführt wird

Das Programm entwickelt sich laufend; der Entwickler arbeitet auf einem **Entwicklungsstand**, der meist **mehrere Versionen vor dem letzten Release** liegt. Dein PR beruht aber auf dem Release. Deshalb ist das Zusammenführen kein einfacher Klick auf **Merge**:

1. **Prüfung:** Der Entwickler (meist mit Hilfe einer KI) vergleicht deine Dateien mit dem Release, auf dem sie beruhen. Das sind genau **deine Änderungen**. Er prüft sie auf Richtigkeit und auf Verträglichkeit mit dem Rest des Programms, auch mit den Regeln oben.
2. **Zusammenführen (3-Wege-Vergleich):** Deine Änderungen werden mit dem **aktuellen Entwicklungsstand** zusammengeführt. Das geht automatisch, wenn dieselben Zeilen nicht anders geändert wurden. Bei einem **Konflikt** löst der Entwickler ihn von Hand, und er fragt bei dir nach, wenn er nicht sicher ist, was du gemeint hast.
3. **Nachbessern:** Möglicherweise bittet er dich, etwas anzupassen: eine Versionsmarke, einen Text, einen Test. Antworte im PR; solange der PR offen ist, bleibt die Diskussion beisammen.
4. **Version und Dokumentation:** Der Entwickler erhöht die Version des Programms, schreibt die Änderung in die Änderungsliste (**HOC**), passt Anleitung und Checkliste an und nennt **dich als Urheber**.
5. **PR schließen:** Weil die Änderung in den Entwicklungsstand übernommen wird (nicht über die GitHub-Schaltfläche), wird der PR danach **mit einem Kommentar geschlossen**, der die Version nennt, in der deine Änderung steckt. Das bedeutet **nicht**, dass sie abgelehnt wurde! Bei einer Ablehnung steht der Grund im Kommentar.
6. **Testen und Release:** Die Betatester prüfen die Änderung (auch auf Systemen, die der Entwickler nicht hat). Dann steht sie im nächsten **Release** und in den **Release-Hinweisen**. Wer vorab testen möchte, nimmt **Hilfe → Update Testversion …**.

Dein Beitrag steht danach im Programmstand und in der Versionsgeschichte. Der Beitrag von *moosbewohner* (Doppelklick in der Spalte **Typ**) ist ein Beispiel dafür, wie das abläuft.

Mit einer KI mitentwickeln

Du musst nicht alles selbst schreiben: Eine KI wie Claude kann Code lesen, ändern und prüfen. Sie ist aber **nur so gut wie ihr Briefing**: Eine neue Sitzung kennt weder das Projekt noch seine Regeln.

So gehst du vor:

1. Hole dir die **neueste Release** von GitHub (das ZIP `webmobaledlib_Dateien_Stand_x.yz.zip` bzw. den Ordner `python`) und die Dateien `HOC_intern.md` und `KI_prompt/KI_ENTWICKLUNG_Prompt.md`.
2. Starte eine neue Sitzung (zum Beispiel in einem Claude-Projekt). Füge den **Prompt unten** als erste Nachricht oder Projektanweisung ein. Hänge die Dateien an.
3. Beschreibe dein Anliegen in einfachen Sätzen („Wenn ich … tue, passiert …, es sollte …“).
4. Lass die KI **erst die Ursache suchen und einen Plan nennen**, dann ändern.
5. **Prüfe das Ergebnis selbst:** Lies die Änderung, probiere sie aus, und lass dir sagen, was die KI **nicht geprüft** hat. Eine KI kann sich irren und behauptet gern, etwas sei getestet.
6. Sende die geänderten Dateien als Pull Request, und schreibe in den PR, dass eine KI mitgeholfen hat.

Prompt für die KI (zum Kopieren):

```
# Auftrag
Du hilfst mir, einen Beitrag (Pull Request) für das Open-Source-Programm
MobaLedLib Studio zu schreiben. Antworte auf Deutsch, sachlich und knapp.
Behaupte nie, etwas sei geprüft, wenn du es nicht ausgeführt hast.
Nenne Unsicheres als unsicher. Stelle höchstens eine Rückfrage auf einmal.

## Das Projekt
MobaLedLib Studio ist die Web-Version des Programmgenerators der MobaLedLib
(LEDs, Servos und Sound für Modellbahnen). Man pflegt in einer Tabelle
Zeilen mit Makros, das Programm erzeugt daraus ein Programm für die
Lichtmaschine (ARDUINO Nano, ESP32 oder Raspberry Pico) und lädt es per USB
hoch. Die Oberfläche läuft im Browser (NiceGUI, Ordner _web). Die Fachlogik
stammt aus dem Excel-Original (VBA) und wurde nach Python übersetzt
(Ordner proggen, pattgen, mlpyproggen, ExcelAPI; headless ersetzt die
Tk-Fenster). Die Datei KI_prompt/KI_ENTWICKLUNG_Prompt.md im Anhang
beschreibt alle Ordner und Fallstricke ausführlich: Lies sie zuerst.
HOC_intern.md im Anhang erklärt, warum die letzten Änderungen gemacht
wurden.

## Meine Aufgabe
(Hier beschreiben: was soll sich ändern, welcher Fehler tritt auf, welche
Meldung kommt, wie stelle ich ihn nach.)

## Vorgehen
1. Lies die betroffenen Dateien und finde die Ursache. Stelle sie nach,
   wenn möglich. Keine Vermutung als Tatsache liefern.
2. Nenne mir kurz deinen Plan, bevor du änderst.
3. Mache kleine, gezielte Änderungen. Kein Umformatieren, keine
   Umbenennungen nebenbei, keine neuen Fremdbibliotheken. Ein Thema pro
   Pull Request. Prüfe, dass neue Namen nicht schon vergeben sind.
4. Regeln für geänderte Python-Dateien:
   - Kopf mit Name, Path (relativ zu python), Version, Last Update,
     Author, Licence LGPL V2.1 und History of Change. Version erhöhen,
     History ergänzen. Fehlt der Kopf: einfügen.
   - Jede geänderte Zeile bekommt die neue Version als Kommentar
     (# v1.15). Zu löschende Zeilen bleiben als Kommentar stehen und
     werden mit der Version markiert.
   - Vorhandene Zeilenenden (CRLF oder LF) behalten.
5. Texte für Anwender sind deutsch und stehen in tr("..."). Der deutsche
   Text ist der Schlüssel in csv/Web_Languages.csv (Trennzeichen ;,
   Spalten: Deutsch, EN, NL, FR, IT, ES, DA). Neue Texte dort ergänzen.
   Namen von Knöpfen, Reitern und Menüs exakt wie im Programm schreiben.
6. Anwender dürfen nie von Hand in wichtigen Dateien editieren müssen.
   Einstellungen gehören als Option in die Oberfläche.
7. Prüfe mit python -m py_compile jede geänderte Datei, führe vorhandene
   und neue Tests aus und prüfe den Start des Programms, soweit in deiner
   Umgebung möglich. Nicht prüfbar sind Browser, Windows, macOS, Raspberry
   Pi und Hardware: Schreibe das ausdrücklich in "Nicht geprüft".
```

```
## Ergebnis
Liefere (a) die geänderten Dateien, (b) eine kurze Liste Datei, Version,
Änderung, (c) eine Beschreibung für den Pull Request mit: Basisversion,
Was und warum, Getestet, Nicht geprüft, Text für die Release-Hinweise
(Neue Funktionen oder Korrekturen), ein Punkt für die Betatest-Checkliste.
Benenne eigene Fehler sofort und behebe sie.
```

Hinweis: Der Prompt ist eine Kurzfassung für Beitragende. Das ausführliche Briefing des Entwicklers liegt als `KI_prompt/KI_ENTWICKLUNG_Prompt.md` im Repository; es enthält außerdem die Arbeitsweise bei Release-Paketen. Die Regeln für Pakete und Releases brauchst du für einen Pull Request **nicht**.

Gut zu wissen

- **Lizenz:** Das Programm steht unter der **LGPL V2.1**. Mit deinem Pull Request erklärst du dich einverstanden, dass dein Beitrag unter dieser Lizenz steht.
- **Keine Passwörter und Zugangsdaten** in Code, Beschreibung oder Anhängen.
- **Geduld:** Der Entwickler arbeitet in seiner Freizeit. Eine Antwort kann dauern; frag ruhig nach, wenn es länger still bleibt.
- **Fragen vorab:** Du bist unsicher, ob deine Idee passt? Lege erst einen Issue an und frage dort.

Siehe auch

- Fehler melden und Wünsche eintragen
- Das Programm aktualisieren (auch die Testversion)
- Begriffe

From:
https://wiki.mobaledlib.de/ - **MobaLedLib Wiki**

Permanent link:
https://wiki.mobaledlib.de/anleitungen/mobaledlibstudio/mitentwickeln

Last update: **2026/11/24 17:11**