

TinyHexa Dokumentation

```
PgUp/PgDn: Scroll  Tab: Switch  F1: Hex/Bin  F2: Save  F3: Reload  F12: Quit

Offset  Hex                               ASCII
00000000 23 20 43 4d 41 4b 45 20 67 65 6e 65 72 61 74 65 # CMAKE generate
00000010 64 20 66 69 6c 65 3a 20 44 4f 20 4e 4f 54 20 45 d file: DO NOT E
00000020 44 49 54 21 0d 0a 23 20 47 65 6e 65 72 61 74 65 DIT!..# Generate
00000030 64 20 62 79 20 22 4d 69 6e 47 57 20 4d 61 6b 65 d by "MinGW Make
00000040 66 69 6c 65 73 22 20 47 65 6e 65 72 61 74 6f 72 files" Generator
00000050 2c 20 43 4d 61 6b 65 20 56 65 72 73 69 6f 6e 20 , CMake Version
00000060 33 2e 33 31 0d 0a 0d 0a 23 20 44 65 66 61 75 6c 3.31...# Defaul

NOTHING changed s8:35 u8:35 s32:1296244771 u32:1296244771
```

Ein Terminal-basierter Hex-/Binär-ASCII-Editor

Studiengang: Elektrotechnik
Fach: Programmieren in C/C++

Autor: Lennart Jaworek
Matrikelnummer: IU14098143
E-Mail: Lennart.Jaworek@iu-study.org
Datum: 21. August 2025

<https://github.com/LeonS1337/TinyHexaIU>

Inhaltsverzeichnis

1 Sinn und Zweck der Software	3
1.1 Beschreibung	3
1.2 Anzeige & Eingabe	3
1.3 Dateihandling	3
1.4 Zielsetzung	4
2 Aktueller Funktionsumfang	4
2.1 Ansichten	4
2.2 Bearbeitung	4
2.3 Navigation und Scrollen	4
2.4 Dateioperationen	5
2.5 Statusleisten	5
2.6 Fehlerbehandlung und Logging	5
2.7 Tastenkürzel (Übersicht)	5
3 Softwarestruktur	6
3.1 Modulübersicht	6
3.2 Datenfluss (kurz)	7
3.3 Eingesetzte Design Pattern	7
3.3.1 MVC	7
3.3.2 Object Pattern (init/deinit)	7
3.3.3 Strategy (Display_Strategy)	8
4 Schnittstellen	9
4.1 Benutzerschnittstelle: Eingaben, Befehle, Parameter	9
4.1.1 Main.c Aufruf (CLI)	9
4.1.2 Main_Controller.c Ereignisverarbeitung (Controller-Loop)	12
4.1.3 editor_input.c Eingaben in Hex/Bin/ASCII	12
4.1.4 Model ↔ File IO: DataBuffer nutzt FileManager	16
5 Kompilierung & Ausführung	19
5.1 Voraussetzungen	19
5.2 Kompilieren mit CMake	20
5.2.1 Unter Windows (getestet auf Windows 11 24H2)	20
5.2.2 Unter Linux (getestet auf Debian 13.0.0)	20
5.3 Ausführung	21
5.4 Kurzanleitung zum Testen	21
6 Abweichungen und Erweiterungen gegenüber dem ursprünglichen Konzept	21
6.1 Verwendung von ncurses statt pdcurses	21
6.2 Umsetzung des Programmablaufs	22

6.3	Reduktion des Funktionsumfangs	22
6.4	Strukturierung mit Design-Patterns	22
6.5	Modulanpassungen	22
7	Einsatz von Künstlicher Intelligenz (KI) im Entwicklungsprozess	22
7.1	Unterstützung bei strukturellen Änderungen (z. B. Strategy Pattern)	23
7.2	Dokumentation und Kommentierung	23
7.3	Fazit	23
8	Aktueller Stand	23
8.1	Bekannte Einschränkungen	23

1 Sinn und Zweck der Software

1.1 Beschreibung

TinyHexa ist ein schlanker, rein tastaturgesteuerter Hex-Editor für das Terminal in C. Er dient dazu, beliebige Dateien zu laden, strukturiert anzuzeigen, direkt bytegenau zu bearbeiten und zu speichern.

1.2 Anzeige & Eingabe

Tabellarische Ansicht mit Offset-Spalte, einer Daten-Tabelle (umschaltbar Hex - Binär, F1) und einer ASCII-Spalte.

PgUp/PgDn: Scroll Tab: Switch F1: Hex/Bin F2: Save F3: Reload F12: Quit

Offset	Hex	ASCII
00000000	23 20 43 4d 41 4b 45 20 67 65 6e 65 72 61 74 65	# CMAKE generate
00000010	64 20 66 69 6c 65 3a 20 44 4f 20 4e 4f 54 20 45	d file: DO NOT E
00000020	44 49 54 21 0d 0a 23 20 47 65 6e 65 72 61 74 65	DIT!..# Generate
00000030	64 20 62 79 20 22 4d 69 6e 47 57 20 4d 61 6b 65	d by "MinGW Make
00000040	66 69 6c 65 73 22 20 47 65 6e 65 72 61 74 6f 72	files" Generator
00000050	2c 20 43 4d 61 6b 65 20 56 65 72 73 69 6f 6e 20	, CMake Version
00000060	33 2e 33 31 0d 0a 0d 0a 23 20 44 65 66 61 75 6c	3.31....# Defaul

NOTHING changed s8:35 u8:35 s32:1296244771 u32:1296244771

Abbildung 1: Tabellarische Ansicht Hex

1.3 Dateihandling

Datei per Kommandozeilenargument oder interaktiver Pfadabfrage öffnen.

```
ljawo | ...\build\MyPreset\src | main | 14:07 | took 0s on LENOVO-LAPTOP
.\P_TinyMiny.exe Makefile
```

Abbildung 2: Kommandozeile Öffnen

Datei schließen und ggf. speichern.

```
ljawo | ...\build\MyPreset\src | main | 14:15 | took 41s on LENOVO-LAPTOP
.\P_TinyMiny.exe Makefile
TinyHexa is running file: Makefile
Would you like to save the file? Press Y to save or N to discard:
```

Abbildung 3: Programm schließen

1.4 Zielsetzung

Einfache, robuste Byte-Inspektion und -Editierung ohne GUI-Overhead.

Im Code: Klare Trennung der Verantwortlichkeiten (MVC-Pattern), austauschbare Darstellungslogik und Erweiterbarkeit (Strategy-Pattern) sowie modularer Aufbau durch Objekte (Object-Pattern).

2 Aktueller Funktionsumfang

2.1 Ansichten

Offset-Spalte, Datentabelle (Hex oder Binär) und ASCII-Spalte.

PgUp/PgDn: Scroll Tab: Switch F1: Hex/Bin F2: Save F3: Reload F12: Quit					
Offset	Bin				ASCII
00000000	00100011	00100000	01000011	01001101	# CM
00000004	01000001	01001011	01000101	00100000	AKE
00000008	01100111	01100101	01101110	01100101	gene
0000000c	01110010	01100001	01110100	01100101	rate
00000010	01100100	00100000	01100110	01101001	d fi
00000014	01101100	01100101	00111010	00100000	le:
00000018	01000100	01001111	00100000	01001110	DO N
NOTHING changed s8:35 u8:35 s32:1296244771 u32:1296244771					

Abbildung 4: Tabellarische Ansicht Bin

Darstellung Hex - Binär umschaltbar (F1).

2.2 Bearbeitung

Hex-Eingabe: zwei Nibbles pro Byte; nach dem zweiten Nibble wird das Byte geschrieben und der Cursor rückt vor.

Binär-Eingabe: acht Bits pro Byte; nach dem 8. Bit wird das Byte geschrieben und der Cursor rückt vor.

ASCII-Eingabe: Zeichen werden direkt als Byte geschrieben; Cursor rückt vor.

Bereichswechsel Tabelle ↔ ASCII per Tab.

2.3 Navigation und Scrollen

Cursor: Pfeiltasten (↑ ↓ ← →).

Seitenweise Scrollen: PgUp / PgDn.

Cursor rutscht am Zeilenende an den Anfang der nächsten; angefangene Eingabesequenzen (Hex/Bin) werden beim Navigieren verworfen.

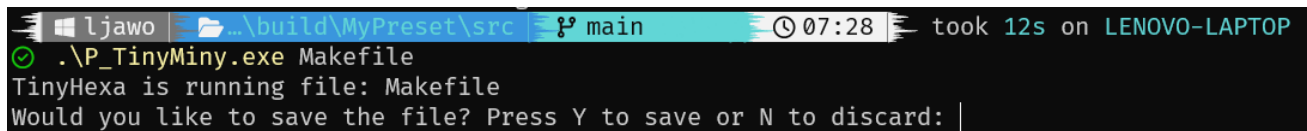
2.4 Dateioperationen

Datei öffnen über Kommandozeilenargument oder interaktive Pfadabfrage.

Speichern in die Quelldatei (F2).

Neu laden/Änderungen verwerfen und Datei erneut laden (F3).

Beenden (F12) beendet die Haupteingabeschleife; außerhalb der TUI folgt eine Abfrage, ob ungespeicherte Änderungen gespeichert werden sollen (Y/N), und ein finaler Status-Text.



```
ljawo | ...\\build\\MyPreset\\src | main | 07:28 | took 12s on LENOVO-LAPTOP
.\P_TinyMiny.exe Makefile
TinyHexa is running file: Makefile
Would you like to save the file? Press Y to save or N to discard: |
```

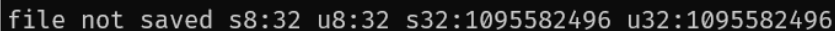
Abbildung 5: Änderungen Speichern

2.5 Statusleisten

Top-Bar: Kurzhinweise „PgUp/PgDn: Scroll Tab: Switch F1: Hex/Bin F2: Save F3: Reload F12: Quit“.

Bottom-Bar:

- Änderungsstatus: „NOTHING changed“ / „file not saved“ / „file saved“.
- Anzeige der Werteinterpretation am Cursor (s8, u8, s32, u32; Little-Endian).



```
file not saved s8:32 u8:32 s32:1095582496 u32:1095582496
```

Abbildung 6: Bottom Bar

2.6 Fehlerbehandlung und Logging

Einheitliche Fehlerausgaben über Err_Log (Standardpfad „log.txt“).

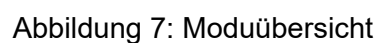
Registrierbare Aufräumfunktion für geordneten Abbruch bei fatalen Fehlern.

Ncurses-Aufrufe sind über Makro abgesichert (Ncurses_Check), um Fehler früh zu erkennen.

2.7 Tastenkürzel (Übersicht)

- F1: Anzeige Hex - Binär umschalten.
- F2: Datei speichern.
- F3: Datei neu laden (verwirft ungespeicherte Änderungen).
- F12: Programm beenden (mit Save-Abfrage außerhalb der TUI).
- Tab: Eingabebereich Tabelle ↔ ASCII umschalten.
- Pfeiltasten: Cursor bewegen.
- PgUp / PgDn: seitenweises Scrollen.

3.1 Modulübersicht



3.2 Datenfluss (kurz)

1. Tastatureingabe → Main_Controller.run() → Auswertung (Pfeile, F-Tasten, Tab).
2. Cursor/Scroll/Moduswechsel → Editor (editor_input.c) → ggf. Strategie-Aufruf.
3. Byte-Schreibzugriff → Data_Buffer → Änderungsflags aktualisieren.
4. Draw-Zyklus → MainWindow → TopBar, Editor (editor_draw.c), Bottom_Bar.

3.3 Eingesetzte Design Pattern

3.3.1 MVC

- Model: Data_Buffer, File_Manager
- View: Main_Window, Top_Bar, Editor (editor_draw, editor_input), Bottom_Bar, Display_Strategy
- Controller: Main_Controller, Utilities, Err_Log
- Verantwortung: Controller verarbeitet Eingaben (Pfeiltasten, PgUp/PgDn, Tab, F1/F2/F3/F12), triggert View-Updates und ruft Model-Operationen auf.

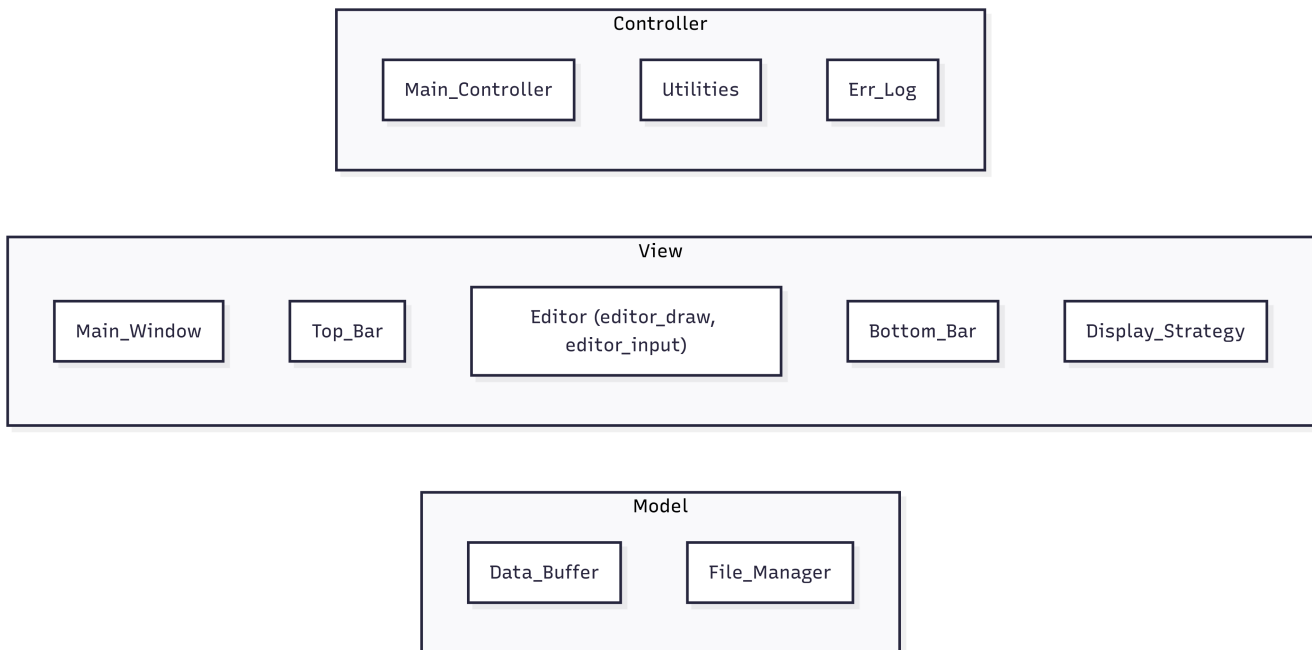


Abbildung 8: MVC Pattern

3.3.2 Object Pattern (init/deinit)

- Jedes „Objekt“ besitzt klar definierte Lebenszyklusfunktionen:
 - init / deinit Funktionen als Konstruktor und Destruktor
- Self-Pointer wird konsequent übergeben; Zuständigkeiten sind jeweils lokal gekapselt.

```
// Beispiele aus Main_Controller.h
```

```
typedef struct {
```



```

    char file_path[256]; /**< Pfad der zu bearbeitenden Datei */
    DataBuffer buffer; /**< Eingelesene Dateidaten */
    MainWindow view; /**< Hauptfenster der Anwendung */
} Main_Controller;

int main_controller_init(Main_Controller *self, const char *file_path);
int main_controller_deinit(Main_Controller *self);

```

3.3.3 Strategy (Display_Strategy)

- Schnittstelle über Funktionszeiger (bytes_per_line, cell_width, is_valid_char, format_byte, reset_pending).
- Konkrete Strategien: HEX_STRATEGY und BIN_STRATEGY.

```

// Beispiele aus Display_Strategy.c

/** Strategie für die Hex-Darstellung. */
const DisplayStrategy HEX_STRATEGY = {
    .bytes_per_line = hex_bytes_per_line,
    .cell_width = hex_cell_width,
    .is_valid_char = hex_is_valid_char,
    .format_byte = hex_format_byte,
    .header_label = hex_header_label,
    .empty_cell = hex_empty_cell,
    .reset_pending = hex_reset_pending
};

/** Strategie für die Binär-Darstellung. */
const DisplayStrategy BIN_STRATEGY = {
    .bytes_per_line = bin_bytes_per_line,
    .cell_width = bin_cell_width,
    .is_valid_char = bin_is_valid_char,
    .format_byte = bin_format_byte,
    .header_label = bin_header_label,
    .empty_cell = bin_empty_cell,
    .reset_pending = bin_reset_pending
};

// Parameterfunktionen
//...
//Bytes pro Zeile in der Hex-Ansicht.

```

```
static int hex_bytes_per_line(void) { return 16; }
//Bytes pro Zeile in der Binär-Ansicht.
static int bin_bytes_per_line(void) { return 4; }
//Überschrift für die Hex-Spalte.
static const char *hex_header_label(void) { return "Hex"; }
//Überschrift für die Binär-Spalte.
static const char *bin_header_label(void) { return "Bin"; }
//...
```

```
// Beispiele aus Display_Strategy.c
```

```
// Struktur mit Funktionszeigern für verschiedene Anzeigeparameter.
typedef struct DisplayStrategy {
    int (*bytes_per_line)(void);           /**< Bytes pro Zeile */
    int (*cell_width)(void);               /**< Zeichenbreite einer Zelle */
    int (*is_valid_char)(int ch);          /**< Prüft Eingabezeichen */
    void (*format_byte)(unsigned char byte, char *out); /**< Formatiert Byte */
    const char *(*header_label)(void);     /**< Überschrift */
    const char *(*empty_cell)(void);       /**< Leere Zelle */
    void (*reset_pending)(struct Editor *self); /**< Eingaben zurücksetzen */
} DisplayStrategy;
```

Der Editor hält einen Zeiger auf die aktive Strategie und wechselt ihn mit `editor_toggle_display_mode()`.

```
//Schaltet zwischen Hex- und Binärdarstellung um.
void editor_toggle_display_mode(Editor *self) {
    //...
    self->strategy = (self->display_mode == DISPLAY_HEX) ?
        &HEX_STRATEGY : &BIN_STRATEGY; // passende Strategie
    //...
}
```

Die aktuelle Strategie liegt im Zeiger `strategy` und stellt virtuelle Funktionen wie `bytes_per_line` oder `format_byte` bereit. Beim Umschalten wird lediglich dieser Strategiezeiger angepasst; Zeichnen und Eingaben nutzen weiterhin dieselbe Editorlogik.

4 Schnittstellen

4.1 Benutzerschnittstelle: Eingaben, Befehle, Parameter

4.1.1 Main.c Aufruf (CLI)

- `tinyhexa <Dateipfad>` — öffnet die angegebene Datei.
- `tinyhexa` (ohne Argument) — interaktive Pfadabfrage in der TUI.

Main.c kümmert sich zunächst nur um die Dateiwahl/Validierung (Argument prüfen, sonst interaktiv pfad anfordern). Erst danach wird der Main_Controller initialisiert und die Hauptschleife gestartet. Ncurses wird hier nicht verwendet – die UI-Initialisierung liegt im View (über den Controller aufgerufen).

```
// Main.c
int main(int argc, char *argv[]) {
    char file_path[256]; // Speicher für den Dateipfad

    // Pfad aus Argumenten verwenden oder vom Nutzer anfordern
    if (argc >= 2 && file_exists(argv[1])) {
        strncpy(file_path, argv[1], sizeof(file_path) - 1); // Argument kopieren
        file_path[sizeof(file_path) - 1] = '\0';           // String terminieren
    } else {
        request_valid_path(file_path, sizeof(file_path)); // gültigen Pfad anfordern
    }

    // Controller anlegen und initialisieren
    Main_Controller controller;
    if (main_controller_init(&controller, file_path) != 0) {
        return 1; // Start fehlgeschlagen
    }

    // Hauptschleife ausführen
    int run_result = main_controller_run(&controller);
    //...
```

Nach Rückkehr aus main_controller_run(...) sichert main zunächst Statusinformationen aus dem Model unsaved_changes und fährt die View herunter. Erst danach fragt main im normalen Terminal (kein ncurses) nach, ob gespeichert werden soll. Anschließend werden Ressourcen freigegeben und ein final Status ausgegeben.

```
// Status des Buffers sichern
// ungespeicherte Änderungen
int unsaved_changes = controller.buffer.edited;
//...

// View herunterfahren
main_window_deinit(&controller.view);

// Offene Änderungen bei Bedarf speichern
if (unsaved_changes) {
    printf("Would you like to save the file? Press Y to save or
    N to discard: ");
```

```

    int answer = getchar();          // erste Eingabe lesen
    getchar();                      // nachfolgendes Newline entfernen
    if (answer == 'y' || answer == 'Y') {
        if (data_buffer_save_file(&controller.buffer, final_path) == 0) {
            unsaved_changes = 0; // Änderungen gespeichert
        }
    }
}

// Buffer freigeben
data_buffer_deinit(&controller.buffer);
memset(&controller, 0, sizeof(controller));

// Abschließenden Status der Datei ermitteln
const char *final_text;
if (!ever_changed) {
    final_text = "NOTHING changed";          // Datei blieb unverändert
} else if (unsaved_changes) {
    final_text = "Last changes DISCARDED";    // Änderungen verworfen
} else {
    final_text = "Last changes SAVED";        // Änderungen gespeichert
}

// Status melden
printf("Final status %s: %s\n", final_path, final_text);

if (run_result != 0) {
    return 1; // Fehlercode weiterreichen
}

return 0; // Erfolg
}

```

Aufrufkette: main → main_controller_init(...) → main_controller_run(...).

UI-Start/Stop passiert in der View über den Controller; main bleibt „sauber“ und kümmert sich um Dateipfad, Prozessstart, Abschluss-Speicherlogik.

Nach Rückkehr aus der UI-Schleife: View deinit (ncurses-Destruktor), dann optionales Speichern via data_buffer_save_file(...), Ressourcen freigeben, finalen Status melden.

4.1.2 Main_Controller.c Ereignisverarbeitung (Controller-Loop)

Eingaben werden zentral vom Controller verarbeitet. Die Haupteingabeschleife läuft, bis F12 gedrückt wird.

```
// Main_Controller.c

//...
while ((input_key = wgetch(self->view.editor.win)) != KEY_F(12)) {
    main_controller_handle_input(self, input_key); // Eingabe verarbeiten
    main_window_draw(&self->view);                // Anzeige aktualisieren
}
//...
```

Berechnung der Seitengröße und Vorbereitung

```
//...
void main_controller_handle_input(Main_Controller *self, int key) {
    // Berechnung der Seitengröße: Zeilen * Bytes pro Zeile
    // nutzbare Zeilen ohne Rahmen und Kopfzeile
    int row_count = getmaxy(self->view.editor.win) - 3;
    // Bytes pro Zeile ermitteln
    int bytes_per_line = getBytesPerLine(&self->view.editor);
    // Bytes, die eine Seite umfasst
    size_t page = (size_t)row_count * (size_t)bytes_per_line;
}
//...
```

Auswahl der Fälle

```
//...
case KEY_UP:    /* Cursor nach oben */
    editor_move_cursor(&self->view.editor, -1, 0); break;
case KEY_DOWN: /* Cursor nach unten */
    editor_move_cursor(&self->view.editor, 1, 0); break;
case KEY_LEFT: /* Cursor nach links */
    editor_move_cursor(&self->view.editor, 0, -1); break
//...
```

4.1.3 editor_input.c Eingaben in Hex/Bin/ASCII

Die eigentliche Interpretation der Tastatureingaben für Hex-, Binär- und ASCII-Modus erfolgt im Editor. Die gezeigten Funktionen liegen in src/View/editor_input.c und bilden die Eingabelogik des Editors ab. Der MainController liest Tastenereignisse (z. B. Tab, F1, Zeichen) und ruft daraufhin diese Editor-Funktionen auf. Änderungen an Bytes werden über data_buffer_set_byte(...) direkt in den DataBuffer (Model) geschrieben; Cursor und sichtbarer Ausschnitt werden im Editor-State aktualisiert.

Beim Umschalten von Bereichen/Modi werden angefangene Eingaben (Hex-/Bit-Puffer) verworfen und der Sichtbereich korrekt neu ausgerichtet.

Bereichswechsel Tabelle ↔ ASCII

Prüft `self->cursor_area` und toggelt zwischen `AREA_HEX` und `AREA_ASCII`. Anschließend ruft sie `clamp_cursor(self)`, damit die Cursorposition im neuen Bereich gültig ist (keine Out-of-Bounds-Koordinaten).

```
// editor_input.c
//...
void editor_toggle_area(Editor *self) {
    if (self->cursor_area == AREA_HEX) {
        self->cursor_area = AREA_ASCII; // in ASCII-Bereich wechseln
    } else {
        self->cursor_area = AREA_HEX;    // zurück zum Hex-Bereich
    }
    clamp_cursor(self); // Cursorposition validieren
}
//...
```

Anzeige Hex - Binär umschalten

- Berechnet den absoluten Byte-Index aus `start_offset`, `cursor_y`, `cursor_x` und `getBytesPerLine(self)`.
- Toggelt `self->display_mode` und setzt `self->strategy` auf `HEX_STRATEGY` oder `BIN_STRATEGY`.
- Setzt angefangene Eingaben zurück (`hex_pending = -1`, `bin_pending_bits = 0`, `bin_pending_value = 0`).
- Ermittelt die Page-Größe aus Fensterhöhe (`getmaxy(self->win) - 3`) und `bytesPerLine`.
- Richtet `start_offset` so aus, dass der absolute Index auf der aktuellen Seite liegt, und berechnet daraus neue `cursor_y/cursor_x`.

```
// editor_input.c
//...
void editor_toggle_display_mode(Editor *self) {
    size_t absoluteIndex = self->start_offset +
        (size_t)self->cursor_y *
        (size_t)getBytesPerLine(self) +
        (size_t)self->cursor_x;
    // aktuelle absolute Position
    self->display_mode = (self->display_mode == DISPLAY_HEX) ?
        DISPLAY_BIN : DISPLAY_HEX; // Modus wechseln
    self->strategy = (self->display_mode == DISPLAY_HEX) ?
        &HEX_STRATEGY : &BIN_STRATEGY; // passende Strategie
    self->hex_pending = -1; // angefangene Hex-Eingabe zurücksetzen
}
```

```

self->bin_pending_bits = 0;    // Bitzähler leeren
self->bin_pending_value = 0;   // Zwischenspeicher löschen
int bytesPerLine = getBytesPerLine(self);
int rows = getmaxy(self->win) - 3;
// Größe einer Seite
size_t pageBytes = (size_t)rows * (size_t)bytesPerLine;
if (absoluteIndex < self->start_offset ||
    absoluteIndex >= self->start_offset + pageBytes) {
    // Seite neu ausrichten
    self->start_offset = (absoluteIndex /
        (size_t)bytesPerLine) * (size_t)bytesPerLine;
}
// relative Position
size_t relative = absoluteIndex - self->start_offset;  auf Seite
self->cursor_y = (int)(relative / (size_t)bytesPerLine);
self->cursor_x = (int)(relative % (size_t)bytesPerLine);
}
//...

```

Hex-Eingabe

- Validiert mit `isxdigit(key)` und wandelt `key` nach `value` (0–15) um.
- Wenn noch kein Nibble aussteht (`hex_pending < 0`): erstes Nibble puffern.
- Sonst: zweites Nibble kombinieren (`(hex_pending << 4) | value`), als Byte via `data_buffer_set_byte(...)` schreiben, `hex_pending` zurücksetzen und `advance_cursor(self)` für die nächste Zelle.

```

// editor_input.c
//...
static void handle_hex_input(Editor *self, int key, size_t index) {
    if (!isxdigit(key)) return; // nur 0-9a-f zulassen
    // Zeichen in Zahl wandeln
    int value = (key <= '9') ? key - '0' : (tolower(key) - 'a' + 10);
    if (self->hex_pending < 0) {
        self->hex_pending = value; // ersten Nibble merken
    } else {
        unsigned char byte = (unsigned char)
            ((self->hex_pending << 4) | value); // Byte bilden
        // Byte in Buffer schreiben
        data_buffer_set_byte(self->buffer, index, byte);
        self->hex_pending = -1; // Zustand zurücksetzen
        advance_cursor(self); // Cursor zur nächsten Position bewegen
    }
}

```

```

    }
}
//...

```

Binär-Eingabe

- Nur '0' oder '1' akzeptieren; bin_pending_value um ein Bit nach links schieben und das neue Bit anhängen.
- bin_pending_bits hochzählen; bei 8 Bits das Byte via data_buffer_set_byte(...) schreiben.
- Nach Commit: Zähler und Zwischenspeicher zurücksetzen und advance_cursor(self) auf die nächste Zelle.

```

// editor_input.c
//...
static void handle_bin_input(Editor *self, int key, size_t index) {
    if (key != '0' && key != '1') return; // nur 0 oder 1 akzeptieren
    // Wert aufbauen: bestehende Bits nach links und neues Bit anhängen
    self->bin_pending_value =
        (unsigned char)((self->bin_pending_value << 1) | (key - '0'));
    self->bin_pending_bits++; // Bitzähler erhöhen
    if (self->bin_pending_bits == 8) { // vollständiges Byte erreicht
        data_buffer_set_byte(self->buffer, index, self->bin_pending_value);
        self->bin_pending_bits = 0; // Zähler zurücksetzen
        self->bin_pending_value = 0; // Zwischenspeicher leeren
        advance_cursor(self); // Cursor weiter
    }
}
//...

```

ASCII-Eingabe

- Prüft mit isprint(key), ob das Zeichen druckbar ist.
- Schreibt das Zeichen als Byte via data_buffer_set_byte(...) in den Buffer und bewegt den Cursor mit advance_cursor(self).

```

// editor_input.c
//...
static void handle_ascii(Editor *self, int key, size_t index) {
    if (isprint(key)) { // nur druckbare Zeichen akzeptieren
        // Byte schreiben
        data_buffer_set_byte(self->buffer, index, (unsigned char)key);
        advance_cursor(self); // Cursor weiterbewegen
    }
}

```



```
}  
//...
```

4.1.4 Model ↔ File IO: DataBuffer nutzt FileManager

Dateizugriffe gekapselt in FileManager; DataBuffer bleibt zuständig für Speicherpuffer und Änderungsstatus.

Dateien:

- src/Model/Data_Buffer.c (ruft FileManager)
- src/Model/File_Manager.h/.c (I/O-Hilfsfunktionen)

Der DataBuffer hält die Datei im RAM (Zeiger + Größe + Änderungs-Flags). FileManager führt die eigentlichen Datei-Operationen zustandslos aus.

Typischer Ablauf:

1. Laden: data_buffer_load_file → readFileToBuffer → neuen Puffer übernehmen → Flags zurücksetzen
2. Bearbeiten: data_buffer_set_byte → Byte im RAM schreiben → Änderungs-Flags setzen
3. Speichern: data_buffer_save_file → writeBufferToFile → „unsaved“-Flag löschen

```
// File_Manager.c  
//...  
int readFileToBuffer(const char *path, unsigned char **outBytes, size_t *outSize) {  
    FILE *file = fopen(path, "rb");  
    if (!file) {  
        return -1; // Datei konnte nicht geöffnet werden  
    }  
  
    // Dateigröße ermitteln  
    if (fseek(file, 0, SEEK_END) != 0) {  
        fclose(file);  
        return -1; // Fehler beim Positionieren  
    }  
    long fileSize = ftell(file); // aktuelle Position entspricht Größe  
    if (fileSize < 0) {  
        fclose(file);  
        return -1; // Größe konnte nicht bestimmt werden  
    }  
    rewind(file); // wieder an den Anfang setzen  
  
    unsigned char *buffer = (unsigned char *)malloc(fileSize); // Speicher anfordern  
    if (!buffer) {
```

```

    fclose(file);
    return -1; // kein Speicher
}

size_t readBytes = fread(buffer, 1, fileSize, file); // Datei lesen
fclose(file);
if (readBytes != (size_t)fileSize) {
    free(buffer); // unvollständig gelesen, Speicher freigeben
    return -1;
}

*outBytes = buffer; // gelesene Bytes zurückgeben
*outSize = readBytes; // gelesene Größe setzen
return 0; // Erfolg
}
//...

```

Eine komplette Datei binär in einen frisch allokierten Heap-Puffer lesen.

- Öffnet die Datei (`fopen(..., "rb")`); bei Fehler → -1.
- Ermittelt die Größe (`fseek/ftell`), springt zurück an den Anfang (`rewind`).
- Allokiert genau passende Buffergröße (`malloc(fileSize)`).
- Liest die Daten vollständig (`fread(..., fileSize)`); bei Teillesung → Puffer freigeben, -1.
- Bei Erfolg: setzt `*outBytes` und `*outSize`, gibt 0 zurück.

```

// Data_Buffer.c
//...
// Datei in Puffer laden
int data_buffer_load_file(DataBuffer *self, const char *path) {
    unsigned char *temp_bytes = NULL; // temporärer Zeiger für gelesene Daten
    size_t temp_size = 0;             // Größe der gelesenen Datei

    // Datei über den File Manager laden
    // Prüfen, ob Lesen klappt
    if (readFileToBuffer(path, &temp_bytes, &temp_size) != 0) {
        // Bei Fehler Programmabbruch und Logeintrag
        // Fehler melden und Programm beenden
        fatal_error("data_buffer_load_file", path);
    }

    // Vorherigen Inhalt freigeben und neue Daten übernehmen
    data_buffer_deinit(self); // alten Puffer leeren
}

```

```

self->bytes = temp_bytes; // neuen Zeiger übernehmen
self->size = temp_size;   // neue Größe setzen
self->edited = 0;         // Bearbeitungsstatus zurücksetzen
self->ever_changed = 0;   // bisherige Änderungen zurücksetzen
return 0; // Erfolg melden
}
//...

```

Den aktuellen RAM-Puffer durch den Dateinhalt ersetzen.

- Ruft readFileToBuffer auf. Bei Fehler → fatal_error("data_buffer_load_file", path) (Log + geordneter Programmabbruch).
- Bei Erfolg:
 - Gibt alten Puffer frei (data_buffer_deinit(self)).
 - Übernimmt den neuen Zeiger und die neue Größe: self->bytes = temp_bytes; self->size = temp_size;.
 - Setzt Flags zurück: self->edited = 0; self->ever_changed = 0;.

```

// Data_Buffer.c
//...
// Puffer sichern
int data_buffer_save_file(DataBuffer *self, const char *path) {
    if (!self->bytes) { // prüfen, ob überhaupt Daten vorhanden sind
        // Kein Inhalt vorhanden, wird als schwerwiegender Fehler behandelt
        // Hinweis auf fehlenden Inhalt
        fatal_error("data_buffer_save_file", "no bytes");
    }
    // schreiben
    int result = writeBufferToFile(path, self->bytes, self->size);
    if (result != 0) { // prüfen, ob Schreiben fehlgeschlagen ist
        // Fehler protokollieren und Programm beenden
        fatal_error("data_buffer_save_file", path); // Fehler beim Schreiben
    }
    self->edited = 0; // Änderungen als gespeichert markieren
    return 0;        // Erfolg melden
}
//...

```

Den Puffer dauerhaft in die Datei schreiben.

- Kein Puffer → fatal_error("data_buffer_save_file", "no bytes").
- Ruft writeBufferToFile(path, self->bytes, self->size) auf.
 - Bei Fehler → fatal_error("data_buffer_save_file", path) (keine Scheinsicherheit).
 - Bei Erfolg → self->edited = 0 (RAM und Datei wieder synchron).

- Keine stillen Fehler; bei I/O-Problemen klarer Abbruch mit Log.

```
// Data_Buffer.c
//...
void data_buffer_set_byte(DataBuffer *self, size_t index, unsigned char value) {
    // prüfen, ob der Zugriff zulässig ist
    if (!self->bytes || index >= self->size) {
        return; // bei Fehler früh beenden
    }
    // Nur reagieren, wenn sich der Wert wirklich ändert
    if (self->bytes[index] == value) { // prüfen, ob neuer Wert identisch ist
        return; // keine Änderung nötig
    }
    self->bytes[index] = value; // neues Byte schreiben
    self->edited = 1; // Puffer als geändert markieren
    self->ever_changed = 1; // Merken, dass jemals etwas geändert wurde
}
//...
```

Ein einzelnes Byte im RAM ändern und Flags setzen.

- Prüft Grenzen und Null-Puffer; bei Fehler sicheres Return.
- Schreibt nur, wenn sich der Wert wirklich ändert (spart unnötige Arbeit/Flags).
- Setzt danach Flag: `edited = 1` (aktuell ungespeichert), `ever_changed = 1` (irgendwann geändert).
- UI kann „unsaved“ korrekt anzeigen; Sichern erfolgt später via Save.

5 Kompilierung & Ausführung

5.1 Voraussetzungen

Betriebssystem: Die Ausführung des Programms ist unter Linux und Windows möglich. Unter Windows wird zum Kompilieren ein POSIX-kompatibles Umfeld wie MinGW empfohlen.

- Datei-I/O läuft über `fopen/fread/fwrite/fclose` (stdio). Diese Funktionen sind auf Linux und Windows identisch verfügbar.
- Die Pfade kommen vom Benutzer (Terminal Argument oder Eingabe). Es gibt im Code keine hard-codierten, systemabhängigen Verzeichnisse.
- Linux: meist `#include <ncurses.h>`; Linkerflag `-lncurses`.
- Windows (MinGW/MSYS2/WSL): üblicherweise `ncurses` aus dem Paketmanager. Header ist oft `#include <ncurses/ncurses.h>` (je nach Paket), Linkerflag `-lncurses`.

Compiler: GCC mit C17-Standard.

Bibliotheken: Eine Besonderheit besteht beim Einbinden der `ncurses`-Bibliothek. Unter Linux trägt sie den Namen `ncurses`, während unter Windows üblicherweise das äquivalente Projekt `PDCurses` verwendet

wird. Alternativ kann auch unter Windows direkt ncurses verwendet werden – vorausgesetzt, es wird ein POSIX-nahes Umfeld wie MSYS2 mit dem UCRT64-Compiler genutzt und ncurses dort als Bibliothek installiert.

Build-Tools: Empfohlen wird CMake ab Version 3.15 als komfortables, plattformübergreifendes Build-System. Als Build-Generator wird Ninja verwendet.

5.2 Kompilieren mit CMake

Für reproduzierbare Builds unter Linux und Windows eignet sich CMake besonders gut. Die ncurses-Bibliothek wird unter windows durch ncurses statisch gelinkt, sodass sie vollständig in die ausführbare Datei eingebettet ist. Auch ein Build unter Windows – mit Visual Studio Code, dem UCRT64-GCC-Compiler und der CMake-Erweiterung ist möglich.

5.2.1 Unter Windows (getestet auf Windows 11 24H2)

Installation MSYS2

https://github.com/msys2/msys2-installer/releases/download/2024-12-08/msys2-x86_64-20241208.exe

Dem Installationsassistenten folgen, anschließend MSYS2 starten.

In MSYS2 die mingw-64 Toolchain installieren.

```
pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain
```

In das Verzeichnis C:\msys2\ wechseln und ucrt64.exe starten.

```
pacman -S cmake
```

```
pacman -S ninja
```

Ncurses ist bereits Teil der Toolchain und benötigt keine zusätzliche Installation.

Zum Kompilieren:

```
cd /c/TinyHexaProjektFolder/  
cmake -G Ninja CMakeLists.txt  
ninja
```

Das erzeugte Binary liegt danach unter \src\TinyHexa.exe.

5.2.2 Unter Linux (getestet auf Debian 13.0.0)

Installation der Build-Tools (GCC, Make etc.)

```
sudo apt update  
sudo apt install -y build-essential
```

Installation von CMake und Ninja

```
sudo apt install -y cmake
sudo apt install -y ninja-build
```

Installation der ncurses-Bibliothek

```
sudo apt install -y libncurses-dev
```

CMake mit Ninja als Build-System verwenden

```
cmake -G Ninja CMakeLists.txt
```

Build-Prozess mit Ninja starten

```
ninja
```

Das erzeugte Binary liegt danach unter `/src/TinyHexa`.

5.3 Ausführung

Unter Linux

```
./tinyhexa <Datei>
```

Unter Windows

```
.\tinyhexa.exe <Datei>
```

Wird keine Datei angegeben, fragt TinyHexa interaktiv nach einem Pfad.

5.4 Kurzanleitung zum Testen

1. Mit einer Beispiel-Datei starten, z.B. `./src/tinyhexa Makefile`.
2. Navigation per Pfeiltasten, Wechsel zwischen Daten- und ASCII-Spalte mit `Tab`.
3. `F1` schaltet zwischen Hex- und Binärdarstellung um.
4. Änderungen mit `F2` speichern, `F3` lädt die Datei erneut.
5. Beenden über `F12`.

6 Abweichungen und Erweiterungen gegenüber dem ursprünglichen Konzept

Im Verlauf der Entwicklung von TinyHexa ergaben sich mehrere Änderungen und Erweiterungen gegenüber der ursprünglichen Projektplanung.

6.1 Verwendung von ncurses statt pdcurses

Ursprünglich war geplant, unter Windows die Bibliothek PDCurses zu verwenden. Während der Entwicklungsarbeiten stellte sich jedoch heraus, dass im MinGW- bzw. MSYS2-Umfeld (insbesondere im UCRT64-Target) eine voll funktionsfähige Variante von ncurses verfügbar ist. Diese bietet den Vorteil einer besseren Kompatibilität mit dem Build-System und ermöglicht plattformübergreifendes Verhalten.

Aufgrund dieser Vorteile wurde ncurses anstelle von pdcurses verwendet, wodurch die Codebasis identisch unter Linux und Windows nutzbar blieb. Pfadangaben und andere plattformspezifische Unterschiede wurden bewusst vermieden, um die Portabilität zu gewährleisten.

6.2 Umsetzung des Programmablaufs

Die ursprünglich im Ablaufdiagramm definierten Phasen Startup, Active und End wurden funktional vollständig umgesetzt, jedoch leicht anders strukturiert.

Startup und Endphase erfolgen direkt in der main.c, wo die Initialisierung der Umgebung sowie die abschließende Freigabe der Ressourcen erfolgt. Die Active-Phase ist im Main_Controller gekapselt.

Diese Trennung erwies sich als praxistauglich und übersichtlich.

6.3 Reduktion des Funktionsumfangs

Während der Umsetzung wurde deutlich, dass der ursprünglich geplante Funktionsumfang sehr ambitioniert war. Aus diesem Grund wurden einzelne Features, z.B. die ASCII-Suche und eine Hilfefunktion, vorerst nicht implementiert. Die zentralen Funktionen wurden jedoch erfolgreich realisiert.

6.4 Strukturierung mit Design-Patterns

Das ursprünglich vorgesehene MVC-Pattern wurde im Verlauf der Implementierung grundsätzlich beibehalten, stieß jedoch in reinem C auf praktische Grenzen – insbesondere bei dem Anspruch, View und Controller vollständig voneinander zu entkoppeln. Eine hybride Struktur, bei der View und Controller zusammen integriert sind und gemeinsam mit dem Model interagieren, hätte sich im Nachhinein möglicherweise als praktikabler erwiesen. Das zugrunde liegende Objekt-Pattern hingegen erwies sich als robust und ließ sich innerhalb der Projektarchitektur gut umsetzen.

Ein wichtiger struktureller Meilenstein war die Einführung des Strategy-Patterns zur Laufzeitumschaltung zwischen Hex- und Binärdarstellung. Dieses wurde erst im späteren Projektverlauf eingeführt, ersetzte jedoch erfolgreich eine Vielzahl an bedingten if-Verzweigungen und trug entscheidend zur Lesbarkeit und Wartbarkeit des Codes bei.

6.5 Modulanpassungen

Die im Konzept vorgesehene Modulübersicht wurde im Wesentlichen umgesetzt. Einige Module wie z. B. ein separates Exception Handling erwiesen sich in der Praxis als redundant und wurden zugunsten eines zentralen Fehlerprotokolls im Err_Log zusammengeführt. Diese Konsolidierung reduzierte die Komplexität ohne funktionalen Verlust.

7 Einsatz von Künstlicher Intelligenz (KI) im Entwicklungsprozess

Zu Beginn der Projektarbeit bestand das Ziel darin, die Entwicklung möglichst ohne KI-gestützte Unterstützung umzusetzen. Der Fokus lag auf einer eigenständigen, manuellen Umsetzung, um ein vertieftes

Verständnis für die in C umgesetzten Konzepte zu erlangen.

Im weiteren Verlauf der Implementierung wurde jedoch deutlich, dass der gezielte Einsatz von KI-gestützten Tools – insbesondere GitHub Copilot – erhebliche Vorteile im Hinblick auf Effizienz, Fehlervermeidung und strukturelle Codequalität bieten kann. So wurde GitHub Copilot zunehmend als Assistenzsystem in den Entwicklungsprozess integriert.

7.1 Unterstützung bei strukturellen Änderungen (z. B. Strategy Pattern)

Ein besonders markanter Nutzen zeigte sich bei der Umstellung auf das Strategy Pattern, welches im späteren Projektverlauf eingeführt wurde, um eine Vielzahl von if-Verzweigungen für die Darstellung zwischen Hex- und Binärformat abzulösen. Die KI unterstützte hierbei nicht nur bei der strukturellen Umstellung des Codes, sondern auch beim Refactoring vorhandener Funktionen und der Definition der passenden Funktionsschnittstellen. Dies sparte signifikant Entwicklungszeit und trug zu einer klareren, wartbaren Architektur bei.

7.2 Dokumentation und Kommentierung

Ein weiterer Anwendungsbereich der KI lag in der automatisierten Generierung von Kommentaren, insbesondere in Vorbereitung auf eine spätere Dokumentation im Format von Doxygen. Für viele Funktionen wurden automatisch sinnvolle Beschreibungen der Parameter, Rückgabewerte und Funktionszwecke generiert. Diese maschinell erstellten Kommentare dienten als Ausgangsbasis für die manuelle Verfeinerung und erleichterten eine konsistente Schnittstellendokumentation.

7.3 Fazit

Der Einsatz von KI wurde im Verlauf der Projektumsetzung nicht nur als akzeptable, sondern als zielgerichtet sinnvolle Ergänzung wahrgenommen. Die KI fungierte dabei nicht als Ersatz für eigenes Verständnis, sondern als Werkzeug zur Effizienzsteigerung und Qualitätssicherung – besonders bei umfangreichen oder sich wiederholenden Programmieraufgaben.

8 Aktueller Stand

Die Software ist aktuell voll funktionsfähig und lauffähig unter den Betriebssystemen Linux und Windows. Sie wurde mit dem Ziel entwickelt, plattformübergreifend einsetzbar zu sein, was durch den Verzicht auf betriebssystemspezifische Pfade sowie durch den gezielten Einsatz von plattformunabhängigen Bibliotheken erreicht wurde. Die Software verhält sich bei typischer Nutzung fehlerfrei und kann daher als releasefertig in Version 1.0.0 betrachtet werden.

8.1 Bekannte Einschränkungen

Aktuell existieren folgende Einschränkungen in der Funktionalität:

- Kein Unicode-Support (UTF-8): Die Anzeige beschränkt sich auf einfache ASCII-Zeichen. Eine Darstellung von Unicode-Inhalten ist derzeit nicht möglich, da ncurses standardmäßig keine UTF-8-Ausgabe unterstützt. Die Integration einer UTF-8-fähigen Variante – wie etwa ncursesw – wäre jedoch mit überschaubarem Aufwand realisierbar und stellt eine mögliche Erweiterung dar. Diese ist jedoch noch nicht getestet.
- Keine Suchfunktion / Kein Direktsprung zu Offsets: Eine Funktion zur gezielten Suche nach Byte-Folgen oder ASCII-Strings sowie eine Sprungmöglichkeit zu bestimmten Speicheradressen ist bisher nicht implementiert. Gerade beim Bearbeiten großer Dateien würde dies den Bedienkomfort erheblich verbessern.
- Bottom-Bar: begrenzte Datendarstellung: In der aktuellen Version zeigt die untere Leiste bereits einige Interpretationen des aktuellen Bytes an. Die Erweiterung um weitere Datentypen oder Darstellungsmöglichkeiten ist jedoch technisch ohne größeren Aufwand möglich. Ein zukünftiges Optionsmenü zur Konfiguration der Anzeige wäre an dieser Stelle sinnvoll.
- Fehlende Fensteranpassung: Die Anwendung erfordert eine Mindestgröße des Terminalfensters. Ist diese unterschritten, startet das Programm nicht. Auch bei dynamischer Änderung der Fenstergröße während der Laufzeit erfolgt derzeit keine automatische Anpassung der Darstellung, was in einem zukünftigen Release verbessert werden soll.
- Darstellungsfehler bei SSH-Nutzung in Powershell: Bei Verwendung über eine SSH-Verbindung treten in der Top-Bar kleinere Anzeigefehler auf, insbesondere bei der Umrandung. Die Funktionalität bleibt hiervon unberührt.
- Derzeit wird die Eingabe im Hex- oder Binärbereich nicht sofort visuell aktualisiert. Die Anzeige erfolgt erst, nachdem die Eingabe vollständig abgeschlossen wurde. Besonders bei der binären Eingabe wäre jedoch ein direktes visuelles Feedback wünschenswert. Für zukünftige Versionen ist daher geplant, die Eingabe über ein kleines Overlay-Fenster oder Popup umzusetzen, das über dem jeweiligen Eingabefeld erscheint. Aus diesem Grund wurde die sofortige Aktualisierung bislang bewusst weggelassen.