ED: Editor in C und Win-API

Inhaltsverzeichnis

1 Vorwort

Willkommen zu ED, einem der kleinsten Windows-Editoren der Welt! ED wurde rein in C und Win-API ohne weitere Bibliotheken wie MFC etc. entwickelt. Er ist wie NotePad ein Editor für Windows, beherrscht aber zusätzliche Funktionen wie einen Vollbildmodus auf Knopfdruck, einen Text-Zoom, einen Nur-Lese-Modus, die Verkleinerung in die Systemstatusleiste, unterschiedliche Textfarben etc.

Die nachfolgenden Sektionen dokumentieren Anwendung und Entwicklung von ED.

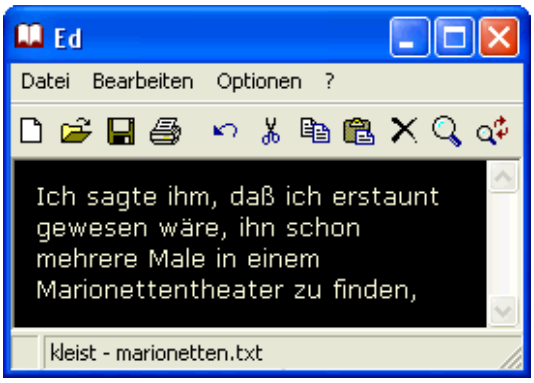
Nutzungsvermerk

2 Dokumentation für Anwender

2.1 Wozu ein weiterer Editor?

ED entstand als Teilkomponente eines größeren Projektes. Gesucht wurde ein Texteditor unter Windows auf Basis der Programmiersprache C und der Win-API unter Ausschluß systemfremder Bibliotheken. Der Editor sollte literarische Texte anzeigen und editieren können und offen für Erweiterungen des Quelltextes sein.

Nun stellt ja schon Windows einen solchen bereit (Editor bzw. NotePad), jedoch ohne Quelltext. Im Internet hingegen finden sich zwar Text-Prozessoren mit deutlich größerem Funktionsumfang; keiner aber erfüllte in toto die Anforderungen an die gesuchte Komponente.



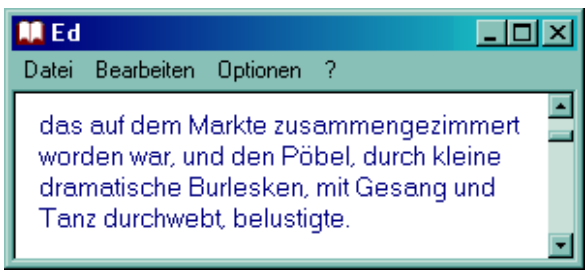
Da bei anderen Editoren immer das eine oder andere fehlte, kam es zur Eigenentwicklung. Zugegebenermaßen reizte auch die Möglichkeit, mit viel Spaß wieder einmal im reinen C zu programmieren - ohne überdimensionierte und komplexe Bibliotheken, nur mit den Windows-Funktionen und einem Compiler. Resultat dieses Ansatzes sind insgesamt ca. 1.600 Quellzeilen, die kompiliert unter LCC einen Editor ergeben, der - je nach gesetzter Compilerdirektive - zwischen 17 und 24 KB groß (oder besser: klein) geraten ist.

Als Dokumentenformat verwendet ED reinen Text, der als einfachstes Format andere Dokumentenformate wie HTML, PDF, DOC, RTF etc. lange überdauern wird und somit Investitionsschutz bietet: auch in vielen Jahren werden Textdateien noch les- und schreibbar sein; ein komplexes und damit epochenabhängiges Anzeigeprogramm wird man auch zukünftig nicht benötigen. (Wer kann hingegen heute noch Wordstar-Dateien anzeigen?) Zudem benötigt man in der Phase der Texterstellung keine weitere Funktionalität.

In einem Roman finden sich auch nur vereinzelt Schriftauszeichnungen; zumeist werden nur Absätze und ggf. Einrückungen verwendet. Die ENTER-Taste in ED beendet somit konsequent den Absatz und nicht die einzelne Zeile. Die sparsame Verwendung von Formatierung führt den lesenden Blick wieder auf das Wesentliche, den reinen Text, zurück und lenkt nicht mit Formatspielereien ab. Manchem mag dies spartanisch vorkommen; für den Projektzweck reichte es vollauf.

2.2 Funktionsumfang von ED

ED ähnelt funktional dem Editor des Betriebssystems. Textzeilen werden allerdings immer am Fensterrand umgebrochen, da das Programm zum Editieren und Lesen von Prosa und nicht zum Bearbeiten von Quelltexten konzipiert wurde. Für letztere finden sich weitaus besser geeignete Programmeditoren im Netz.



Wie beim Editor des Betriebssystems können Textdateien auf fünf unterschiedlichen Wegen geladen werden: beim Aufruf von ED als Parameter auf der Kommandozeile, innerhalb von ED über Menübefehle oder Tastenkürzel, durch das Ziehen einer Textdatei auf die ED-Programmdatei in einem Ordner oder in das offene Fenster von ED. Beginnt eine Datei im Text mit **`.LOG`**, fügt ED beim Laden das aktuelle Datum ein, um Protokolle zu ermöglichen. Bearbeitet werden können nur ANSI-, keine UNICODE-Dateien. Eine entsprechende Erweiterung war nicht geplant, da keine exotischen Zeichensätze unterstützt werden müssen, läßt sich aber bei Bedarf schnell einbinden.

In Ergänzung zum Betriebssystemeditor bietet ED weitere Funktionen:

- Vollbildmodus (F11)
- Programmverkleinerung in die Systemstatusleiste (ESC)
- Wechsel des Farbschemas, z.B. zwischen Tag und Nacht (F2)
- Zoomen des Textes (Strg-+)
- Einstellung von Schrift- und Hintergrundfarbe
- separate Einstellung der Druckschrift
- Drucken markierten Textes
- Optionale Werkzeugleiste
- Nur-Lese-Modus
- einstellbarer Abstand zwischen Fensterrand und Inhalt

Weitere Funktionen können bei Bedarf schnell eingebaut werden.

2.3 Tastatur- und Maussteuerung

Tastatur- und Maussteuerung von ED lehnen sich an die des Betriebssystem-Editors an. Zusätzlich stehen folgende Befehle bereit:

2.3.1 Tastatur

`F2` schaltet das Farbschema um. Dies kann bei wechselnden Lichtverhältnissen (Tag/Nacht) sehr hilfreich sein (z.B. dunkler Text auf hellen Hintergrund oder umgekehrt). Hat der Benutzer noch keine eigene Farben festgelegt, invertiert ED die aktuelle Vorder- und Hintergrundfarbe.

Zur Vergrößerung resp. Verkleinerung des Textes muß die Taste **`Strg`** zusammen mit der Taste **`Plus`** oder **`Minus`** des Ziffernblockes (Numpad) gedrückt werden.

`F11` schaltet zwischen Voll- und Normalbild um.

Mit **`Esc`** flüchtet ED in die Systemstatusleiste (System Tray), mit Maus-Doppelklick auf sein Symbol kommt er wieder zum Vorschein.

Tastenkürzel von ED

Kürzel	Befehl	Kürzel	Befehl	Kürzel	Befehl
Strg-N	Neue Datei	Strg-Einf	Kopieren	Strg-Num+	Schrift vergrößern
Strg-O	Datei öffnen	Strg-V	Einfügen	Strg-Num-	Schrift verkleinern

Kürzel	Befehl	Kürzel	Befehl	Kürzel	Befehl
Strg-S	Datei schließen	Shift-Einf	Einfügen	F1	Hilfe
Strg-P	Datei drucken	Entf	Löschen	F2	Farbmodus
Alt-F4	Beenden	Strg-F	Suchen	Esc	Flüchten
Alt-Rück	Rückgängig	F3	Weitersuchen		
Strg-Z	Rückgängig	Strg-R	Ersetzen		
Strg-X	Ausschneiden	Strg-A	Alles markieren		
Shift-Entf	Ausschneiden	Strg-D	Datum einfügen		
Strg-C	Kopieren	F11	Vollbild an/aus		

Anmerkung: Drückt man erst **F11** und dann **Esc**, passiert beim Reaktivieren von ED etwas lustiges: Das Fenster erscheint zwar mit dem enthaltenen Text, aber ohne Titel-, Menü- und Statusleiste (Post-It-View wie bei Notepad Plus). Tastaturkurzkommandos (Shortcuts wie z.B. **Strg-N**: Neue Datei) funktionieren zwar noch, Menüzugriffskommandos (Access Keys wie z.B. **Alt-D-N**: Neue Datei) aber nicht mehr. Mit **F11** läßt sich der Zustand wieder beheben und das Programm wandert wieder in eine voll funktionale Normalansicht. „It's not a bug, it's a feature."

2.3.2 Maus

Die Maussteuerung von ED entspricht der des Betriebssystemeditors. Ein Rechtsklick offeriert ein Kontextmenü.

2.4 Systemvoraussetzungen

Benötigt wird das Windows-Betriebssystem. Getestet wurde die Einsatzfähigkeit von ED unter Windows 2000, XP, 8 und 10 (ED-Version 1.2 ab XP). Dies schließt nicht aus, daß ED unter anderen Versionen des Betriebssystems installierbar ist, wurde aber nicht überprüft.

2.5 Installation und Deinstallation

Im Gegensatz zu Windows-Programmen, die sich mehr oder weniger fest mit dem System verdrahten, erzeugt ED keine Änderungen an Systemdateien (Registry etc.). Das bedeutet, zur Installation ist ED nur in einen Ordner zu kopieren, z.B. in

```
C:\Programme\ed\
```

Falls der Benutzer Einstellungen bleibend speichern will, erzeugt ED noch eine Konfigurationsdatei namens ed.ini. Zur Deinstallation von ED reicht die Löschung von ed.exe und (ggf.) ed.ini.

Bis Version 1 schrieb ED die Konfiguration einfach in seinen Programmordner, wie es auch viele ältere Programme taten. Unter jüngeren Windows-Systemen mit Benutzerkontensteuerung (UAC), also ab Windows Vista, sind jedoch zumeist Schreibrechte auf den Programmordner notwendig, die man als Benutzer standardmäßig nicht hat. Um dennoch älteren Programmen das Schreiben von Anwendungsdaten zu ermöglichen, virtualisiert das Betriebssystem den Ort der Konfigurationsdatei, so daß sich diese effektiv in einem anderen Ordner befindet. Windows 8 lenkt z.B. nicht-administrative Schreibzugriffe eines (fiktiven) Benutzers Tim auf den Programmordner

```
C:\Program Files (x86)\ed\
```

transparent auf den Virtualisierungsordner

```
C:\Users\Tim\AppData\Local\VirtualStore\Program Files (x86)\ed\
```

um.

Da die Virtualisierung die verwendeten Pfade noch unübersichtlicher macht, verfolgt ED ab Version 1.1 folgende Strategie, um dem Benutzer das Speichern seiner Konfiguration zu ermöglichen: Will der Benutzer seine Konfiguration speichern, sucht ED zuerst im Programmordner seine Konfigurationsdatei, um einen portablen Betrieb zu ermöglichen; findet ED eine, schreibt es die Konfiguration dort hinein. Findet es keine, speichert ED diese unterhalb des vom Betriebssystem dafür vorgesehenen Anwendungsdatenordners ab, z.B. für Windows 8 unter

```
C:\Users\Tim\AppData\ed\
```

oder unter Windows XP unter

```
C:\Dokumente und Einstellungen\Tim\Anwendungsdaten\ed\
```

Umgekehrt versucht ED die Konfiguration zuerst aus seinem Programmordner zu lesen und nur bei Fehlen dieser aus dem Anwendungsdatenordner. Dies ermöglicht einen portablen Betrieb, z.B. vom USB-Stick. Soll nicht im Anwendungsdatenordner gespeichert werden, reicht es, manuell eine leere Datei namens ed.ini im Programmordner von ED zu erzeugen.

Ab Version 1.2 befindet sich ein Manifest mit einer UAC-Direktive in ED, so daß Windows den Programmzugriff auf Pfade nicht grundsätzlich virtualisiert.

2.6 Konfigurationsdatei

Speichert der Benutzer Einstellungen in ED, legt dieser in seinem Programmverzeichnis eine Datei ed.ini an, um sich die getroffenen Einstellungen zu merken. Alle Einstellungen finden sich als Einträge in ed.ini unter der Rubrik [Options] und können auch direkt (ohne ED) geändert werden. Sollte ED nach direkten Änderungen an ed.ini nicht mehr funktionieren, kann die Datei ed.ini einfach gelöscht werden; Ed startet dann wieder mit seinen fest eingetragenen Vorgabewerten.

Konfigurationseinträge von ed.ini

Eintrag	Beispielwert	Kommentar
Window	-	Fensterkoordinaten
ScrFontName	Verdana	Bildschirmschriftart
ScrFontSize	-21	Bildschirmschriftgröße
ScrFontWeight	400	Bildschirmschriftdicke
PrtFontName	CG Times	Druckschriftart
PrtFontSize	-16	Druckschriftgröße

Eintrag	Beispielwert	Kommentar
PrtFontWeight	400	Druckschriftdicke
ScrFontColor1	00000	Bildschirmschriftfarbe
ScrBgColor1	E1FFE1	Bildschirmhintergrundfarbe
ScrFontColor2	FFFFDD	Alternative Bildschirmschriftfarbe
ScrBgColor2	00000	Alternative Bildschirmhintergrundfarbe
ToolBar	0	Werkzeugleiste an (1) oder aus (0)
StatusBar	0	Statusleiste an (1) oder aus (0)
Padding	10	Abstand Fensterinhalt zum -Rahmen

Anmerkungen: Der Abstand des Fensterinhaltes zum Rahmen (Padding) läßt sich unter ED nicht ändern und kann nur in ed.ini direkt eingetragen werden. Farbangaben folgen der hexadezimalen RGB-Notation (RRGGBB). Ein reines Blau als Beispiel ist also mit 0000FF zu erzeugen.

3 Dokumentation für Entwickler

3.1 Entwicklungsgrundlage

ED basiert nur auf der Programmiersprache C, auf der Win-API und setzt keine MFC o.ä. Bibliotheken voraus. Somit sollten die Quelltexte mit jedem C-Compiler problemlos übersetzt werden. Danken möchte ich an dieser Stelle den zahllosen Autoren im Internet, deren Ansätze mich bei der Entwicklung von ED inspiriert haben.

Namen eigener Funktionen und Variablen fangen zur besseren Unterscheidung von Windows-Routinen mit kleinem Buchstaben an. Entstammen sie einem anderen Modul, sind sie zudem mit Modulkürzel und Unterstrich präfigiert, um die Herkunft anzuzeigen. Die Anzahl globaler Variablen wurde auf das Mindestmaß reduziert, Seiteneffekte weitestgehend vermieden. Zur besseren Lesbarkeit wurden weitere funktionale Erweiterungen aus den Quelltexten entfernt. Das Programm läßt sich sowohl als ANSI- als auch UNICODE-Variante übersetzen. Weitere Funktionsbausteine sind variabel über Compiler-Direktiven eingebunden.

Die eigentliche Editierlogik liegt nicht im Quellcode, sondern in der Edit-Klasse des Betriebssystems. Wer also Quelltexte zu Zeilenpuffern oder abstrakte Datentypen zur Textverwaltung sucht, wird hier nicht fündig. Der gesamte Quelltext behandelt die Fensterverwaltung und in der EDIT-Klasse nicht enthaltene Funktionen wie Suchen, Ersetzen, Druck etc.

3.2 Kompilation

Die Quelltexte können sowohl innerhalb der integrierten Entwicklungsumgebungen als auch auf der Kommandozeile übersetzt werden. Wer noch keinen Compiler sein eigen nennt, kann einen solchen im Netz frei und kostenlos herunterladen. Beispielhaft sei nur die Kommandozeilenversion BCC32 von Borlands Entwicklungsumgebung, die Express-Edition von Visual C sowie der LCC- oder MingW-Compiler genannt. Getestet wurde der Quelltext von ED unter den Compilern (in Klammern die Größe des resultierenden Kompilats von ED Version 1) BCC32 V5.5.1 (69 KB), LCC V3.8 (23 KB), MSC V15.0 / Visual C 2008 (67 KB) und Pelles C für Windows V7.00.355 (50 KB).

Wer keine Lust auf komplexe Entwicklungsumgebungen hat oder aber nur über einen älteren Rechner verfügt, kann mit den o.g. Befehlen auf der Kommandozeile den Editor selbst kompilieren. Hierfür sind entweder die Programmpfade von Compiler, Linker und Resourcen-Compiler im Programmsuchpfad (path) einzutragen oder beim Aufruf anzugeben. Zudem muß in den nachstehenden Beispielen für **<INC>** bzw. **<LIB>** der Pfad zu den Include-Dateien bzw. Bibliotheken der entsprechenden Entwicklungsumgebung eingetragen werden, sofern er der Umgebung nicht z.B. über Umgebungsvariablen (INCLUDE, LIB) o.ä. bekannt ist. Ab ED Version 1.2 wird die Bibliothek comctl.dll V6 benötigt, eine Kompilierung unter Windows 2000 ist nur mit einem Handstand möglich. Falls ein Pfad Leerzeichen enthält, ist er in Anführungszeichen einzufassen. Die Groß- und Kleinschreibung der aufgeführten Schalter und evt. Leerzeichen dazwischen müssen zumeist beachtet werden. So muß z.B. für

```
brcc32 -i<INC> ed.rc
```

wenn die Include-Dateien im Verzeichnis c:\meine programme\bc\include liegen, konkret

```
brcc32 -i"c:\meine programme\bc\include" ed.rc
```

geschrieben werden.

Beispiel einer Kompilierung unter Borland C V5.5.1:

```
> brcc32 -i<INC> ed.rc
> bcc32 -I<INC> -c -d ed.c fr.c st.c ut.c
> ilink32 -L<LIB> -c -aa -Gn c0w32+ed+fr+st+ut, ed, nul,
import32.lib+cw32.lib+comctl32.lib,, ed.res
```

Beispiel einer Kompilierung unter LCC V3.8:

```
> for %i in (*.c) do lcc %i
> lrc ed.rc
> lcclnk -s ed.obj ut.obj fr.obj st.obj ed.res shell32.lib
```

Anmerkung: LCCs C-Compiler V3.8 Build Nov. 24 2011 initialisiert das Toolbarbutton-Array zum Kompilationszeitpunkt leider fehlerhaft, LCC 3.8 Build Jul 30 2006 kompiliert das Array hingegen korrekt. Umgekehrt kompiliert LCCs Resourcen-Compiler aus dem Buld 2006 keine VERSION_INFO, dafür aber LRC aus dem Build 2011! Wer unter LCC partout alles braucht, kann entweder das Array zur Laufzeit definieren, oder die Quellen unter Build 2006 compilieren und die Resource mit einem anderen Compiler erstellen.

Beispiel einer Kompilierung unter LCC V3.8 mit Compiler-Driver:

```
> lc ed.c fr.c st.c ut.c ed.rc shell32.lib -subsystem windows -s
```

Beispiel einer Kompilierung unter Visual C 2008:

```
> rc /i<INC1>;<INC2> ed.rc
> cl /I<INC1> /I<INC2> /GF ed.c fr.c st.c ut.c ed.res /link
/LIBPATH:<LIB1> /LIBPATH:<LIB2> user32.lib gdi32.lib comdlg32.lib
comctl32.lib shell32.lib
```

Beispiel 1 einer Kompilierung unter Pelles C V7.00.355:

```
> for %i in (*.c) do pocc /I<INC1> /I<INC2> /Ze /Gz /Os %i
> porc /I<INC1> /I<INC2> ed.rc
> polink -LIBPATH:<LIB1> -LIBPATH:<LIB2> user32.lib gdi32.lib comdlg32.lib
comctl32.lib shell32.lib ed.obj ed.res fr.obj st.obj ut.obj
```

Beispiel 2 einer Kompilierung unter Pelles C V7.00.355 mit Compiler-Driver:

```
> cc /Ze /Gz /Os ed.c fr.c st.c ut.c ed.rc user32.lib gdi32.lib
comdlg32.lib comctl32.lib shell32.lib
```

Hinweis: Ein Browser kann die obigen Kommandozeilen am Fensterrand umbrechen. Wer die Aufrufe selbst in der Konsole nachvollziehen möchte, sollte diese Umbrüche nicht als Zeilenende interpretieren.

3.3 Entwicklungsgeschichte

3.3.1 Version 1.2

- Manifest ergänzt, um die Virtualisierung des Betriebssystems zu verhindern
- Visuelle Stile (Dialoge, Toolbars etc.) implementiert (Windows-Versionen ab XP bieten sowohl comctl.dll V.5 (Standard) als auch V.6 an; nur letztere liefert visuelle Stile, die mit einem Manifest und InitCommonControlsEx() aktiviert werden)
- Benutzerkontensteuerung ergänzt (UAC im Manifest)
- Kompatibilitätsmodus ergänzt (Manifest)
- Anpassung auf hochauflösende Ausgabegeräte (Manifest)
- Dialogzeichensatz für 64-Bit-Versionen auf Segoe UI gesetzt
- Quelltext erweitert für Kompilierung mit 64 bit
- Laden und Speichern der Fensterposition

3.3.2 Version 1.1

- Konfiguration nun entweder im Programm- oder Anwendungsdaten-Ordner
- Beginnt eine Datei im Text mit `.LOG`, fügt ED beim Laden das aktuelle Datum ein
- SaveAs setzt nun korrekt Fensterstatus einer schreibgeschützten Datei auf R/W
- Beim Laden einer R/O-Datei wird nun auch Fensterstatus auf R/O gesetzt
- Programmsymbole verbessert

3.3.3 Version 1.0

- Quelltext-Annotation und Vereinheitlichung mit Function-Docs
- Umschreibung Quelltext-Kommentare in Deutsch
- Prüfung UNICODE/ANSI-Kompatibilität und unterschiedlicher Direktiven
- Nachtmodus (alternatives Farbschema)
- Option Druck von Markierungen
- Option Druck von Seitennummern
- Verbesserung Druckrandausgleich für Proportionalschriften
- Menüpunkt Seite einrichten
- Druckränder
- Quelltextzerlegung in Module, Reduktion redundanter oder globaler Variablen
- Übernahme Markierung in Suche
- CAN_UNDO nun auch für INSDATE und ReplaceText(SendMessage(hWinEdit, EM_REPLACESEL, 0->1, (LPARAM)s))
- Programmminimierung auf Anforderung in Systemstatusleiste
- Stabilisierung der Suche bei gleichzeitigen Änderungen am Text
- Wortweise Suche
- Suche unter Berücksichtigung von Groß- und Kleinschreibung
- Erhöhtes Textmaxlimit bei RICHEDIT und EDIT
- Ersatz generischer RTL-Stringfunktionen (_tcs*()) durch Windowsfunktionen (lstrcpy() etc.). Jedoch: nicht alle RTL-Funktionen haben ein Windows-Pendant
- Drag and Drop von Textdateien auf das Programm innerhalb der Shell
- Kommandozeile in UNICODE
- Berücksichtigung quotierter Dateinamen in Kommandozeile
- Suchen/Ersetzen

- Behandlung von WM_QUERYENDSESSION als Windows-Shutdown-Nachricht
- Vergrößertes Textpadding
- Skalierung von Text (Ctrl+/-)
- Datumseinfügung in Text
- Drag and Drop von Textdateien in offenes Editorfenster
- Auslesen von Text aus Statusleiste
- Vollbildmodus
- Optionaler Nur-Lese-Modus
- Erhöhung der erlaubten Editiergröße
- Speicherung von Einstellung in Konfigurationsdatei
- Konditional Bereitstellung von Menübefehlen
- Kommando Alles markieren
- Statusmeldungen
- Ausblendbare Werkzeug- und Statusleiste
- Frei auswählbare Text- und Hintergrundfarbe
- Frei auswählbare Textgröße
- Prototyp

3.3.4 Offen

- Druckabbruch-Routine
- Lesezeichen

3.3.5 Zurückgestellt

- Verarbeitung von UNICODE-Textdateien; bisher nicht benötigt
- Aufbau FR-Puffer nur bei Bedarf; Pufferaufbau macht sich nur bei ReplaceAll zeitlich spürbar bemerkbar. Gerade hier aber muß Puffer neu angelegt werden, da der Inhalt sich geändert hat. Interessanterweise erhöht sich Prg-Speicherbedarf lt. TaskMan nicht – wohl weil OS vorsorglich den in FileLoad() allozierten Puffer intern noch nicht freigegeben hat. Daher erst einmal auf weitere Optimierung verzichtet
- Auslagerung Farbsetzungen in Funktionen und Fontsetzung bei RichEdit über RichEdit-Nachricht mit Farbe
- Internationalisierung; bisher nicht benötigt
- Nur-Lese-Modus auf andere Hintergrundfarbe; bisher erfolglos
- Bei RichEdit 1.0 müßte EN_UPDATE für Parent angemeldet werden; bisher nicht benötigt
- Bei RichEdit wird schon beim initialen Textladen Control „dirty“; bisher nicht benötigt

3.4 RichEdit

Der eine oder andere wird nachstehend noch nützliche Hinweise zur Aufrüstung auf RichEdit finden. RichEdit bringt manches an Funktionalität schon eingebaut mit, was das EDIT-Control noch schmerzlich vermissen läßt, z.B. eine eingebaute Suchen- und Ersetzen-Funktion oder der direkte Druck aus dem Control.

Insbesondere für das Suchen und Ersetzen von Text können zwar die selbstgebauten Routinen auch für RichEdit genutzt werden; aus Performanzgründen sollten aber den RichEdit-eigenen Routinen der Vorzug gegeben werden, da hier nicht für jeden Suchvorgang ein eigener Puffer aufgebaut werden muß.

RichEdit ist konzeptuell zwar grundsätzlich schon im Quelltext hinterlegt, bedarf aber noch einer Konsolidierung, da sich diese Windows-Klasse je nach Versionsstand in Teilaspekten unterschiedlich verhält. Einige dieser Besonderheiten, die in der Entwicklung auffielen, sind nachstehend aufgeführt.

3.4.1 Besonderheiten

- RICHEDIT setzt Farben anders (initial EM_SETBKGNDCOLOR, EM_SETCHARFORMAT) als EDIT (Behandlung von WM_CTLCOLOREDIT)
- RICHEDIT bietet im Gegensatz zu EDIT eine eigene Suchen- und Ersetzen-Routine
- RICHEDIT und EDIT sind standardmäßig auf 64/32 KB beschränkt und verfügen über unterschiedliche Nachrichten zur Heraufsetzung der Speichergrenze
- EN_UPDATE: Muß für V1 via EM_SETEVENTMASK angemeldet werden; V2 hingegen liefert auch ohne Anmeldung EN_UPDATE an das Elternfenster – leider wird aber schon das initiale Laden des Textes als Änderung interpretiert
- V2 blendet VSCROLLBAR aus, wenn Text klein genug
- EM_SETRECT/EM_GETRECT wirken kumulativ, das Padding nimmt bei jeder Größenänderung des Fensters zu
- Statt "RICHEDIT", "RICHEDIT20A", "RICHEDIT20W" etc. sollte beim Laden der RichEdit-Bibliothek immer RICHEDIT_CLASS angegeben werden. Dies garantiert immer den richtigen Klassennamen unabhängig von der Kompilation als UNICODE- oder ANSI-Variante

3.4.2 RichEdit-Bibliotheken und zugehörige Klassennamen

- Version 1.0; DLL: riched32.dll; Windows-Klasse: RICHEDIT_CLASS
- Version 2.0; DLL: riched20.dll; Windows-Klasse: RICHEDIT_CLASS
- Version 3.0; DLL: riched20.dll; Windows-Klasse: RICHEDIT_CLASS
- Version 3.1; DLL: riched20.dll; Windows-Klasse: RICHEDIT_CLASS
- Version 4.1; DLL: msftedit.dll; Windows-Klasse: MSFTEDIT_CLASS
- Version 8.0; DLL: msftedit.dll; Windows-Klasse: MSFTEDIT_CLASS

3.4.3 Vom Betriebssystem unterstützte RichEdit-Versionen

- Windows 8: RichEdit 8.0, RichEdit 3.1
- Windows 7: RichEdit 4.1, RichEdit 3.1
- Windows Vista: RichEdit 4.1, RichEdit 3.1 und ein RichEdit-1.0-Emulator
- Windows XP SP1: RichEdit 4.1, RichEdit 3.0 und ein RichEdit-1.0-Emulator
- Windows XP: RichEdit 3.0 und ein RichEdit-1.0-Emulator
- Windows Me: RichEdit 3.0 und 1.0
- Windows 2000: RichEdit 3.0 und ein RichEdit-1.0-Emulator
- Windows NT 4.0: RichEdit 2.0 und 1.0
- Windows 98: RichEdit 2.0 und 1.0
- Windows 95: Nur RichEdit 1.0. Jedoch ist riched20.dll kompatibel zu Windows 95 und kann durch eine Anwendung bereits installiert sein

Anmerkung: Hier sind nur in Consumer-Betriebssystemen enthaltene RichEdit-Versionen aufgeführt. Server-Systeme, Office-Pakete oder Pocket-Systeme können andere Versionen mitbringen, z.B. Office 2007 mit RichEdit Version 6.0.

3.5 Quelltexte

3.5.1 ed.c

```
/** Editor ED - Modul ED: Editor in C und Win-API **/

// Cop. 2008, 2017 www.asdala.de. Alle Rechte vorbehalten.

/** Include-Dateien **/
#include "top.h" // Compile-Scope, muss zuerst kommen
#include <windows.h>
#include <shlobj.h> // SHGetFolderPath etc. V1.1 New
#include <commctrl.h> // SB_SETPARTS etc.
#ifdef BUILD_RE
#include <richedit.h>
#endif
#include "etc.h" // Compiler-spez. Ergaenzungen
#include "res.h" // Ressourcen-Konstanten
#include "ut.h" // Utility-Funktionen
#ifdef BUILD_FR
#include "fr.h" // Suchen/Ersetzen-Funktionen
#endif
#ifdef BUILD_ST
#include "st.h" // Systemtray-Funktionen
#endif

#ifdef UNICODE
#pragma message ("UNICODE-Version")
#else
#pragma message ("ANSI-Version")
#endif
#ifdef _WIN64
#pragma message ("WIN64-Version")
#else
#pragma message ("WIN32-Version")
#endif

/** Globale Variablen **/
static const TCHAR appTitle[] = TEXT("ED");
static TCHAR filePath[MAX_PATH]; // Von diversen Funktionen via ofn gesetzt
static const TCHAR fileDefExt[] = TEXT("txt"); // Standarddateierweiterung
static const TCHAR fileFilter[] =
  TEXT("Texte (*.txt, *.log)\0*.txt;*.log\0Alle Dateien (*.*)\0*.*\0");
static int dirty, readOnly, printPgNo=1; // Datei veraendert, schreibgeschuetzt, Drucke Seitenzahl
#ifdef BUILD_RE
static HINSTANCE richEditLib; // RichEditbibliothek
#endif

/** Praeprozessordirektiven **/
#define MAXTEXT 200000 // Max. Textgroesse von EDIT/RICHTEXT-Control
#define MESTAT(val) ((val) ? MF_ENABLED : MF_GRAYED) // Schalte Menueeintraege ein oder aus
enum { STSTAT, STINDI, STFILE }; // Namen der Statusleistenteile statt 0, 1, 2

/** Function: Ermittele HDC des Standarddruckers **/
static HDC getDefaultPrinterDC(void)
 {
  PRINTDLG pd;
  ZeroMemory(&pd, sizeof(PRINTDLG));
  pd.lStructSize = sizeof(PRINTDLG);
  pd.Flags = PD_RETURNDEFAULT | PD_RETURNDC;
  PrintDlg(&pd);
```

```
  if (pd.hDevMode)
    GlobalFree(pd.hDevMode); // Falls beim Aufruf von PrintDlg hDevMode und hDevNames NULL sind,
  if (pd.hDevNames) // alloziert OS Speicher und fuellt Strukturen mit Druckerwerten;
    GlobalFree(pd.hDevNames); // daher sollten wir Speicher wieder freigeben.
  return pd.hDC;
  }

/** Function: Menue Waehle Schrift aus **/
static BOOL menuOptionsFont(HWND hWnd, LOGFONT *lf, DWORD cfFlags)
  {
  CHOOSEFONT cf;
  BOOL result;
  ZeroMemory(&cf, sizeof(CHOOSEFONT));
  cf.lStructSize = sizeof(CHOOSEFONT);
  cf.hwndOwner = hWnd;
  if (cfFlags & CF_PRINTERFONTS) // menuOptionsFont() fuer Druckschrift aufgerufen
    if ((cf.hDC=getDefaultPrinterDC()) == NULL) // Nur Schriften des Standarddruckers verfuegbar
      return FALSE;
  cf.lpLogFont = lf; // Druck- oder Screen-Logfont
  cf.Flags = CF_INITTOLOGFONTSTRUCT | cfFlags; // Kein CF_EFFECTS, da Underlined etc. unnoetig
  result = ChooseFont(&cf); // Mit CF_INITTOLOGFONTSTRUCT befuellt ChooseFont indirekt auch logFont
  if (cf.hDC)
    DeleteDC(cf.hDC);
  return result;
  }

/** Function: Lade ANSI-Datei ins EDIT-Fenster **/
static BOOL fileLoad(HWND hWnd, TCHAR fileName[])
  {
  HANDLE hFile;
  BOOL ok = FALSE;
  DWORD fileSize, read;
  char *buf;

  hFile = CreateFile(fileName, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_EXISTING, 0, NULL);
  if (hFile != INVALID_HANDLE_VALUE) {
    fileSize = GetFileSize(hFile, NULL);
    if (fileSize != 0xFFFFFFFF) {
      if ((buf = (char*)GlobalAlloc(GPTR, fileSize + 1)) != NULL) {
        if (ReadFile(hFile, buf, fileSize, &read, NULL)) { // Es gibt keine *A- bzw. *W-Version
          buf[fileSize] = '\0'; // [ANSIONLY]; Null im Text interpretiert EDIT-Control als Textende
          ok = SetWindowTextA(hWnd, buf); // [ANSIONLY]
          }
        GlobalFree(buf);
        }
      }
    CloseHandle(hFile);
    }
  return ok;
  }

/** Function: Schreibe EDIT-Fenster in ANSI-Datei **/
static BOOL fileSave(HWND hWnd, TCHAR fileName[])
  {
  HANDLE hFile;
  BOOL ok = FALSE;
  DWORD textLen, bufSize, written;
  char *buf;

  hFile = CreateFile(fileName, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
```

```
  if (hFile != INVALID_HANDLE_VALUE) {
    if ((textLen=GetWindowTextLength(hWnd)) > 0) {
      bufSize = textLen + 1;
      if ((buf =(char*)GlobalAlloc(GPTR, bufSize)) != NULL) {
        if (GetWindowTextA(hWnd, buf, bufSize)) // [ANSIONLY]
          ok = WriteFile(hFile, buf, textLen, &written, NULL);
        GlobalFree(buf);
        }
      }
    CloseHandle(hFile);
    }
  return ok;
  }

/** Function: Menue Datei Neu **/
static void menuFileNew(HWND hWnd)
  {
  HWND hWinEdit = GetDlgItem(hWnd, IDW_WIN_EDIT);
  SetDlgItemText(hWnd, IDW_WIN_EDIT, TEXT("")); // Loesche EDIT-Text
  SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STINDI, (LPARAM)TEXT("")); // Oder "Neu"
  SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STFILE, (LPARAM)TEXT("")); // Oder "Unbenannt"
  dirty = 0;
  *filePath = 0;
  SendMessage(hWinEdit, EM_SETREADONLY, readOnly=0, 0);
  }

/** Function: Lade Datei **/
static BOOL fileOpen(HWND hWnd, TCHAR newFilePath[], int userReadOnly)
  {
  HWND hWinEdit = GetDlgItem(hWnd, IDW_WIN_EDIT);
  TCHAR s[6] = TEXT("\4");
  int i;
  readOnly = userReadOnly; // V1.1 Fix
  #ifdef BUILD_RO
  if (!readOnly)
    readOnly = (GetFileAttributes(newFilePath) & FILE_ATTRIBUTE_READONLY) ? 1 : 0;
  #endif
  if (!fileLoad(hWinEdit, newFilePath)) {
    ut_messageBoxExt(hWnd, TEXT("Kann Datei '%s' nicht laden!"), appTitle, MB_ICONWARNING,
    newFilePath);
    return FALSE;
    }
  lstrcpyn(filePath, newFilePath, MAX_PATH);
  SendMessage(hWinEdit, EM_SETREADONLY, readOnly, 0);
  SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STINDI, (LPARAM)TEXT("")); // Oder "Geoeffnet"
  SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STFILE, (LPARAM)filePath);
  dirty = 0;
  ShowWindow(hWnd, SW_SHOW);
  /* V1.1: Steht ein .LOG zu Beginn der Datei, fuege Datum am Ende ein */
  if (!readOnly)
    if (SendMessage(hWinEdit, EM_GETLINE, (WPARAM)0, (LPARAM)s) > 0)
      if (lstrcmp(s,TEXT(".LOG")) == 0) {
        i = GetWindowTextLength(hWinEdit);
        SendMessage(hWinEdit, EM_SETSEL, (WPARAM)i, (LPARAM)i);
        SendMessage(hWinEdit, EM_REPLACESEL, (WPARAM)FALSE, (LPARAM)TEXT("\r\n--- "));
        SendMessage(hWnd, WM_COMMAND, (WPARAM)IDM_EDIT_INSDATE, (LPARAM)0);
        SendMessage(hWinEdit, EM_REPLACESEL, (WPARAM)FALSE, (LPARAM)TEXT("\r\n"));
        }
  return TRUE;
  }
```

```
/** Function: Menue Datei oeffnen **/
static void menuFileOpen(HWND hWnd)
  {
  OPENFILENAME ofn;
  ZeroMemory(&ofn, sizeof ofn);
  ofn.lStructSize = sizeof ofn;
  ofn.hwndOwner = hWnd;
  ofn.lpstrFilter = fileFilter;
  ofn.lpstrFile = filePath;
  ofn.nMaxFile = MAX_PATH;
  ofn.lpstrDefExt = fileDefExt;
  ofn.Flags = OFN_FILEMUSTEXIST;
  if (!GetOpenFileName(&ofn))
    return;
  fileOpen(hWnd, filePath, ofn.Flags & OFN_READONLY); // Merke Benutzerwahl RO oder RW aus Dialog
  }

/** Function: Menue Datei speichern unter **/
static void menuFileSaveAs(HWND hWnd)
  {
  HWND hWinEdit;
  OPENFILENAME ofn;
  ZeroMemory(&ofn, sizeof ofn);
  ofn.lStructSize = sizeof ofn;
  ofn.hwndOwner = hWnd;
  ofn.lpstrFilter = fileFilter;
  ofn.lpstrFile = filePath;
  ofn.nMaxFile = MAX_PATH;
  ofn.lpstrDefExt = fileDefExt;
  ofn.Flags = OFN_PATHMUSTEXIST | OFN_OVERWRITEPROMPT;
  if (!GetSaveFileName(&ofn))
    return;
  hWinEdit = GetDlgItem(hWnd, IDW_WIN_EDIT);
  if (!fileSave(hWinEdit, filePath)) {
    ut_messageBoxExt(hWnd, TEXT("Kann Datei '%s' nicht speichern!"), appTitle, MB_ICONWARNING,
    filePath);
    return;
    }
  SendMessage(hWinEdit, EM_SETREADONLY, readOnly=0, 0); // V1.1 Fix
  SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STINDI, (LPARAM)TEXT(""));
  SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STFILE, (LPARAM)filePath);
  dirty = 0;
  }

/** Function: Menue Datei speichern **/
static void menuFileSave(HWND hWnd)
  {
  HWND hWinEdit = GetDlgItem(hWnd, IDW_WIN_EDIT);
  if (*filePath == 0) {
    menuFileSaveAs(hWnd);
    return;
    }
  if (!fileSave(hWinEdit, filePath)) {
    ut_messageBoxExt(hWnd, TEXT("Kann Datei '%s' nicht speichern!"), appTitle, MB_ICONWARNING,
    filePath);
    return;
    }
  SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STINDI, (LPARAM)TEXT(""));
  dirty = 0;
```

```
 }

/** Function: Menue Seite einrichten **/
static void menuFilePageSetup(PAGESETUPDLG *psd)
 {
 if (PageSetupDlg(psd)) { // Druckseiteninitialisierung schon in WM_CREATE erfolgt
   if (psd->hDevMode) { // Falls beim Aufruf von PageSetupDlg hDevMode und hDevNames NULL sind,
     GlobalFree(psd->hDevMode); psd->hDevMode = NULL; }// alloziert OS Speicher und fuellt die
   if (psd->hDevNames) { // Strukturen mit Druckerwerten; daher geben wir Speicher wieder frei.
     GlobalFree(psd->hDevNames); psd->hDevNames = NULL; }
   }
 }

/** Function: Drucke Datei **/
static BOOL filePrint(HWND hWnd, PAGESETUPDLG *psd, LOGFONT *lfPrt, PRINTDLG *pd)
 {
 BOOL ok = TRUE;
 DOCINFO di;
 HFONT hSavedPrintFont, hPrintFont;
 HDC hDCScr;
 TEXTMETRIC tm;
 TCHAR *buf;
 SIZE lnSize;
 HWND hWinEdit = GetDlgItem(hWnd, IDW_WIN_EDIT);
 int yChar, linesPerPage, line, page, dpiXPrt, dpiYPrt, dpiYScr, bufLen, i, iSOL, iEOL;
 LONG marginLeft, marginRight, marginTop, marginBottom, hSavedlfHeight, xPage, yPage, xPrint,
 yPrint;

 /* Ermittele DPI von Drucker und Bildschirm */
 hDCScr = GetWindowDC(hWnd);
 dpiYScr = GetDeviceCaps(hDCScr, LOGPIXELSY);
 ReleaseDC(hWnd, hDCScr);
 dpiXPrt = GetDeviceCaps(pd->hDC, LOGPIXELSX);
 dpiYPrt = GetDeviceCaps(pd->hDC, LOGPIXELSY);

 /* Rechne Druckraender von Hundertstel mm in Pixel um */
 marginLeft = psd->rtMargin.left * dpiXPrt / 2540; // 2500 / 100 * 600 dpi / 25,4 mmpi =
 marginTop = psd->rtMargin.top *  dpiYPrt / 2540; // 2500 * 600 / 2540 = 590 px
 marginRight = psd->rtMargin.right * dpiXPrt / 2540;
 marginBottom = psd->rtMargin.bottom * dpiYPrt / 2540;

 /* Waehle Druckschrift ueber (passager korrigierten) LOGFONT aus */
 // Da LOGFONT auf (veralteter) Bildschirmetrik von 72 dpi fusst, ist Druckschriftgroesse (300 dpi
 // etc.) zu korrigieren, sonst Ausdruck zu klein. Zudem muss alte LONGFONT-Groesse des Druckfonts
 // gemerkt werden, da sonst beim naechsten Aufruf von menuOptionsFont() der Dialog die korrig.
 // (z.B. 4fache) Groesse als Startwert darstellen und bei jedem weiteren Aufruf vergroessern
 // wuerde. In den meisten Dokus wird ueber CHOOSEFONT.iPointSize der korrigierte Wert ermittelt.
 // Annahme: Benutzer-Punktgroesse von 10 Pt (= 10*1/72 "), somit iPointSize = 10 x PtSize = 100).
 // Dann gilt: lfHeight = - Punktgroesse * Druck-DPI / Urbildschirm-DPI
 // Berechnung lfHeight vom Font 10 Pt fuer Bildschirm-Uraufloesung (72 dpi, MAC):
 // lfHeight = - 10
 // Berechnung von lfHeight vom Font 10 Pt fuer heutige Bildschirm-Standardaufloesung (96 dpi):
 // lfHeight = - 10 * 96 / 72 = -13,333
 // Berechnung lfHeight vom Font 10 Pt fuer Bildschirm-Hochaufloesung (120 dpi):
 // lfHeight = - 10 * 120 / 72 = -16,666
 // Berechnung lfHeight vom Font 10 Pt fuer Druckeraufloesung 600 dpi:
 // lfHeight = - 10 * 600 / 72 = -83,333
 // Da CHOOSEFONT.iPointSize nur in menuOptionsFont() z.V. steht und das Programmende nicht
 // ueberdauert, berechnen wir korrigiertes lfPrt.lfHeight direkt aus unkorrigiertem mit Faktor
 // DruckDPI (z.B. 600 dpi) / BildschirmDPI (meist 96 dpi):
```

```
// lfPrt.lfHeight = lfPrt.lfHeight * dpiYPrt / dpiYScr
hSavedlfHeight = lfPrt->lfHeight; // Merke alte LOGFONT-Groesse
lfPrt->lfHeight = MulDiv(lfPrt->lfHeight, dpiYPrt, dpiYScr);  // Passagere LOGFONT-Korrektur
hPrintFont = CreateFontIndirect(lfPrt); // Erstelle Druckfont
lfPrt->lfHeight = hSavedlfHeight; // Restauriere alten LOGFONT
hSavedPrintFont = SelectObject(pd->hDC, (HGDIOBJ)hPrintFont);
if (hSavedPrintFont == NULL || hSavedPrintFont == (HGDIOBJ)GDI_ERROR)
  MessageBox(hWnd, TEXT("Kann Druckfont nicht laden!"), appTitle, MB_ICONWARNING);

/* Ermittele Pixel-Metrik der Druckschrift und -seite */
GetTextMetrics(pd->hDC, &tm); // Ermittle Metrik des aktuell phys. Fonts des ausgew. Druckers hDC
yChar = tm.tmHeight + tm.tmExternalLeading; // Druckzeilenhoehe = Druckzeichenhoehe + Durchschuss
xPage = GetDeviceCaps(pd->hDC, HORZRES);
yPage = GetDeviceCaps(pd->hDC, VERTRES);
xPrint = xPage - marginLeft - marginRight; // Bedruckbare Breite [px]
yPrint = yPage - marginTop - marginBottom; // Bedruckbare Hoehe [Rasterlinien]
linesPerPage = yPrint / yChar;

/* Lade EDIT-Control in Puffer */
if ((buf = pd->Flags & PD_SELECTION ? ut_getEditSel(hWinEdit, &bufLen) :
  ut_getEditText(hWinEdit, &bufLen)) == NULL)
  return FALSE; // EM_GETLINE unterscheidet nicht weiche / harte Umbrueche, daher alles kopieren

/* Benenne Druckjob */
ZeroMemory(&di, sizeof di);
di.cbSize = sizeof di;
if (pd->Flags & PD_PRINTTOFILE)
  di.lpszOutput = TEXT("FILE:");
di.lpszDocName = *filePath ? filePath : TEXT("Unbenannt");

/* Drucke */
if (StartDoc(pd->hDC, &di) > 0) {

  /* Fuer jede Druckseite */
  for (page=i=iSOL=0; i<bufLen; page++) {
    if (StartPage(pd->hDC) <= 0) {
      ok = FALSE; break; }

    /* Fuer jede Druckzeile */
    for (line=0; i<bufLen && line<linesPerPage; line++) {

      /* Ermittele Druckzeilenende iEOL */
      for (lnSize.cx=iEOL=0; i<bufLen && lnSize.cx<xPrint; i++) {
        if (buf[i] == TEXT(' '))
          iEOL = i;
        else if (buf[i] == TEXT('-'))
          iEOL = i + 1;
        else if (buf[i] == TEXT('\r')) {
          iEOL = i; i += 2; break; }
        if (!GetTextExtentPoint32(pd->hDC, buf+iSOL, i+1-iSOL, &lnSize)) // Berechne lnSize.cx
          MessageBox(hWnd, TEXT("Druckmetrik nicht bestimmbar.\nPrüfen Sie den Ausdruck!"),
          appTitle, MB_ICONWARNING);
        };
      if (iEOL == 0) // Bei uebergrossen Zeichenketten wuerde kein iEOL gesetzt, daher
        iEOL = i - 1; // zwangsweise hier
      if (i >= bufLen && buf[iEOL] != TEXT('\r')) // Falls letzte Zeile ohne CRLF, wuerde nur bis
        iEOL = bufLen; // vorletzt. Zeichen (1 Wort auf Zeile) oder bis vor letzt. Blank gedruckt

      /* Drucke Druckzeile */
      TextOut(pd->hDC, marginLeft, marginTop+line*yChar, buf+iSOL, iEOL-iSOL); // Drucke Zeile
```

```
      switch (buf[iEOL]) { // Ermittele naechsten Druckzeilenanfang
        case TEXT(' '): iSOL = iEOL + 1; break;
        case TEXT('\r'): iSOL = iEOL + 2; break;
        default: iSOL = iEOL; // Lass iSOL nur Trennzeichen ueberspringen, aber nicht '-' etc.
        }
      } // Fuer jede Druckzeile

    /* Drucke Seitennummer */
    if (printPgNo) {
      TCHAR num[6];
      int numWidth = wsprintf(num, TEXT("%d"), page+1);
      TextOut(pd->hDC, xPage/2, yPage-marginBottom+100, num, numWidth); // Vereinfachte Position
      }
    if (EndPage(pd->hDC) <= 0) {
      ok = FALSE; break; }
    } // Fuer jede Druckseite
  if (ok)
    EndDoc(pd->hDC);
  }

 /* Raeume auf */
 GlobalFree(buf);
 SelectObject(pd->hDC, (HGDIOBJ)hSavedPrintFont);
 DeleteObject(hPrintFont);
 return ok;
 }

/** Function: Menu Datei drucken **/
static void menuFilePrint(HWND hWnd, PAGESETUPDLG *psd, LOGFONT *lf)
 {
 PRINTDLG pd;
 TCHAR saveStatText[200];
 int sos, eos;
 HWND hWinEdit = GetDlgItem(hWnd, IDW_WIN_EDIT);

 /* Praepariere Druckdialog */
 ZeroMemory(&pd, sizeof pd);
 pd.lStructSize = sizeof pd;
 pd.hwndOwner = hWnd;  // Veraltetes PD_PRINTSETUP durch PageSetup ersetzt
 pd.Flags = PD_USEDEVMODECOPIESANDCOLLATE | PD_NOPAGENUMS | PD_RETURNDC;
 pd.nCopies = 1;
 SendMessage(hWinEdit, EM_GETSEL, (WPARAM)&sos, (LPARAM)&eos);
 if (eos > sos) // Falls Text markiert ist,
   pd.Flags |= PD_SELECTION; // schlage "Markierung" als Druckbereich im Druckdialog vor

 /* Rufe Druckdialog auf */
 if (!PrintDlg(&pd))
   return;

 /* Drucke */
 SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_GETTEXT, STINDI, (LPARAM)saveStatText); //  Speichere
 SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STINDI, (LPARAM)TEXT("Drucke..."));
 SetBkMode(pd.hDC, TRANSPARENT);
 if (!filePrint(hWnd, psd, lf, &pd))
   MessageBox(hWnd, TEXT("Kann Datei nicht drucken!"), appTitle, MB_ICONWARNING);

 /* Raeume auf */
 SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STINDI, (LPARAM)saveStatText);
 if (pd.hDevMode)
   GlobalFree(pd.hDevMode); // Falls beim Aufruf von PrintDlg hDevMode und hDevNames NULL sind,
```

```
  if (pd.hDevNames) // alloziert OS Speicher und fuellt Strukturen mit Druckerwerten; daher sollten
    GlobalFree(pd.hDevNames); // wir Speicher wieder freigeben.
  if (pd.hDC)
    DeleteDC(pd.hDC);
  }

/** Function: Rueckfrage zur Dateispeicherung vor Programmende **/
static int fileSaveBeforeExit(HWND hWnd)
  {
  int i = ut_messageBoxExt(hWnd,
    TEXT("Der Text in der Datei '%s' wurde geändert.\nSollen die Änderungen gespeichert werden?"),
    appTitle, MB_ICONQUESTION | MB_YESNOCANCEL, filePath);
  if (i == IDYES)
    menuFileSave(hWnd);
  return i;
  }

#ifdef BUILD_UC

/** Function: Initialisiere EDIT-Farben **/
static void initColors(CHOOSECOLOR *cc, COLORREF *customclrs, HWND hWnd, COLORREF init)
  {
  cc->lStructSize = sizeof(CHOOSECOLOR);
  cc->hwndOwner = hWnd;
  cc->rgbResult = init;
  cc->lpCustColors = customclrs; // Lass Windows die Benutzerfarben merken zwischen Coloraufrufen
  cc->Flags = CC_ANYCOLOR | CC_RGBINIT;
  }

#endif

/** Function: Callback-Funktion des Versionsdialogs **/
static BOOL CALLBACK verDlgProc(HWND hDlg, UINT msg, WPARAM wParam, LPARAM lParam)
  {
  LITEM item;

  switch (msg) {
  case WM_INITDIALOG:
    return TRUE;
  case WM_COMMAND:
    switch (wParam) {
    case IDOK:
      EndDialog(hDlg, TRUE);
      return TRUE;
    case IDCANCEL:
      EndDialog(hDlg, FALSE);
      return TRUE;
    }
  case WM_NOTIFY:
    if (wParam == IDC_VERLINK)
      switch (((LPNMHDR)lParam)->code) {
      // WM_NOTIFY-Meldung SysLink-Element: wParam=IDC_SYSLINK und lParam=NM_CLICK oder NM_RETURN
      case NM_CLICK:
      case NM_RETURN:
        item = ((PNMLINK)lParam)->item;
        // ShellExecute zu Link fkt. nur mit UNICODE gesetzt oder explizit ShellExecuteW().
        // "open" muß ebf. Unicode sein (L""), Resource-Eintrag ist immer in UNICODE
        ShellExecuteW(NULL, L"open", item.szUrl, NULL, NULL, SW_SHOW);
        return TRUE;
      }
```

```
 } // switch(msg)
 return FALSE;
 }

/** Function: Callbackfunktion des Hauptfensters **/
static LRESULT CALLBACK winProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
 {
 static HWND hWinEdit, hWinStat;
 static TCHAR iniPath[MAX_PATH];
 static HFONT hFont;
 static PAGESETUPDLG psd;
 static LOGFONT lfScr, lfPrt;
 #ifdef BUILD_TB
 static HWND hWinTool;
 #endif
 #ifdef BUILD_UC
 static COLORREF bgClr, bgClr1, bgClr2, fgClr, fgClr1, fgClr2;
 static BOOL nightMode;
 static CHOOSECOLOR choosecolor;
 static LOGBRUSH logbrush;
 static HBRUSH hBrush;
 #ifdef BUILD_RE
 static CHARFORMAT cr;
 #endif
 #endif
 static BOOL showWinTool, showWinStat, saveOptions, makeIniDir;
 static int padding;
 #ifdef BUILD_FR
 static UINT fr_msg;
 #endif
 RECT rcWin; // NEU V 1.2.2, um Fenstergroesse von/nach INI speichern zu koennen
 static TCHAR s[MAX_PATH]; // Generisch fuer mehrere Codeabschnitte in winProc verwendet

 switch (msg) {

 case WM_CREATE:
  {
  #ifdef BUILD_TB
  const TBBUTTON tbb[] = {
    // Bei deklarativer Initialisierung wird undokumentiertes, in commctrl.h definiertes,
    // zusaetzliches Paddingfeld benoetigt, da es ein Array ist; bei expl. Initialisierung
    // der einzelnen Felder zur Laufzeit tritt Problem nicht auf.
    // V1.1 Funktioniert nicht mehr mit LCC V3.8 Build 2011
    // StdBitmapID  CommandID       State           Style          Pad  dwData Beschriftung
    { STD_FILENEW,  IDM_FILE_NEW,   TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { STD_FILEOPEN, IDM_FILE_OPEN,  TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { STD_FILESAVE, IDM_FILE_SAVE,  TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { STD_PRINT,    IDM_FILE_PRINT, TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    // { STD_PRINTPRE, IDM_FILE_PRINT, TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { 0,            0,              TBSTATE_ENABLED, TBSTYLE_SEP,    {0}, 0L, 0 },
    { STD_UNDO,     IDM_EDIT_UNDO,  TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { STD_CUT,      IDM_EDIT_CUT,   TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { STD_COPY,     IDM_EDIT_COPY,  TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { STD_PASTE,    IDM_EDIT_PASTE, TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { STD_DELETE,   IDM_EDIT_CLEAR, TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { STD_FIND,     IDM_EDIT_FIND,  TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    { STD_REPLACE,  IDM_EDIT_REPL,  TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    // { 0,            0,              TBSTATE_ENABLED, TBSTYLE_SEP,    {0}, 0L, 0 },
    // { STD_HELP,     IDM_HELP_VER,   TBSTATE_ENABLED, TBSTYLE_BUTTON, {0}, 0L, 0 },
    };
```

```
  TBADDBITMAP tbab;
  #endif
  #ifdef BUILD_UC
  static COLORREF customcolors[16]; // Muss static sein, merkt sich benutzerdef. Farben
  #endif
  const int statwidths[] = {15, 140, -1}; // Li. Statusfeld endet bei 15, mittl. bei 140,
                                          // re. am Fensterende
  /* Lese Ini-Datei aus programPath oder appDataPath */
  lstrcpy(iniPath+GetModuleFileName(NULL,iniPath,MAX_PATH)-3,TEXT("ini"));
  if (!ut_fileExists(iniPath)) { // V1.1 New
    if (SHGetFolderPath(0, CSIDL_APPDATA, 0, 0, iniPath) == S_OK) {
      lstrcat(iniPath, TEXT("\\ed"));
      if (!ut_dirExists(iniPath))
        makeIniDir = TRUE;
      lstrcat(iniPath, TEXT("\\ed.ini"));
      }
    }
  SetRectEmpty(&rcWin); // Setze rcWin auf Default = 0
  GetPrivateProfileStruct(TEXT("Options"), TEXT("Window"), &rcWin, sizeof rcWin, iniPath);
  GetPrivateProfileString(TEXT("Options"), TEXT("ScrFontName"), TEXT("Georgia"),
    lfScr.lfFaceName, 33, iniPath);
  lfScr.lfHeight = GetPrivateProfileInt(TEXT("Options"), TEXT("ScrFontSize"), -20, iniPath);
  lfScr.lfWeight = GetPrivateProfileInt(TEXT("Options"), TEXT("ScrFontWeight"), 0, iniPath);
  GetPrivateProfileString(TEXT("Options"), TEXT("PrtFontName"), TEXT("Georgia"),
    lfPrt.lfFaceName, 33, iniPath);
  lfPrt.lfHeight = GetPrivateProfileInt(TEXT("Options"), TEXT("PrtFontSize"), 0, iniPath);
  lfPrt.lfWeight = GetPrivateProfileInt(TEXT("Options"), TEXT("PrtFontWeight"), 0, iniPath);
  #ifdef BUILD_UC
  GetPrivateProfileString(TEXT("Options"), TEXT("ScrFontColor1"), TEXT(""), s, 7, iniPath);
  fgClr = fgClr1 = s[0] ? ut_revRGBVal(s) : GetSysColor(COLOR_WINDOWTEXT);
  GetPrivateProfileString(TEXT("Options"), TEXT("ScrBgColor1"), TEXT(""), s, 7, iniPath);
  bgClr = bgClr1 = s[0] ? ut_revRGBVal(s) : GetSysColor(COLOR_WINDOW);
  GetPrivateProfileString(TEXT("Options"), TEXT("ScrFontColor2"), TEXT("C3BE98"), s, 7, iniPath);
  fgClr2 = ut_revRGBVal(s);
  GetPrivateProfileString(TEXT("Options"), TEXT("ScrBgColor2"), TEXT("1A0F0B"), s, 7, iniPath);
  bgClr2 = ut_revRGBVal(s);
  #endif
  #ifdef BUILD_TB
  showWinTool = GetPrivateProfileInt(TEXT("Options"), TEXT("ToolBar"), 0, iniPath);
  #endif
  showWinStat = GetPrivateProfileInt(TEXT("Options"), TEXT("StatusBar"), 1, iniPath);
  padding = GetPrivateProfileInt(TEXT("Options"), TEXT("Padding"), 0, iniPath);

  /* Setze Fensterkoordinaten auf vom Benutzer definierte Werte */
  if (rcWin.left)
    MoveWindow(hWnd, rcWin.left, rcWin.top, rcWin.right-rcWin.left, rcWin.bottom-rcWin.top, 0);

  /* Baue Child-Fenster EDIT */
  #ifdef BUILD_RE
  #if BUILD_RE == VER01 // RichEdit 1.0
  if ((richEditLib=LoadLibrary(TEXT("riched32.dll"))) == NULL)
    MessageBox(hWnd, TEXT("Kann riched32.dll nicht laden!"), appTitle, MB_ICONWARNING);
  hWinEdit = CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("RICHEDIT"), NULL,
    WS_CHILD | WS_VISIBLE | ES_MULTILINE | WS_VSCROLL | ES_AUTOVSCROLL | ES_NOHIDESEL,
    0, 0, 0, 0, hWnd, (HMENU)IDW_WIN_EDIT, GetModuleHandle(NULL), NULL);
  #else // RichEdit 2.0 und 3.0
  if ((richEditLib=LoadLibrary(TEXT("riched20.dll"))) == NULL)
    MessageBox(hWnd, TEXT("Kann riched20.dll nicht laden!"), appTitle, MB_ICONWARNING);
  hWinEdit = CreateWindowEx(WS_EX_CLIENTEDGE, RICHEDIT_CLASS, NULL,
    WS_CHILD | WS_VISIBLE | ES_MULTILINE | WS_VSCROLL | ES_AUTOVSCROLL | ES_NOHIDESEL,
```

```
  0, 0, 0, 0, hWnd, (HMENU)IDW_WIN_EDIT, GetModuleHandle(NULL), NULL);
#endif
SendMessage(hWinEdit,EM_EXLIMITTEXT,(WPARAM)0,(LPARAM)(DWORD)MAXTEXT); // RE: Erhoehe 32K Std
#else
hWinEdit = CreateWindowEx(WS_EX_CLIENTEDGE, TEXT("EDIT"), NULL, // Init. Text _innerhalb_ EDIT
  WS_CHILD | WS_VISIBLE | ES_MULTILINE | WS_VSCROLL | ES_AUTOVSCROLL | ES_NOHIDESEL,
  0, 0, 0, 0, hWnd, (HMENU)IDW_WIN_EDIT, GetModuleHandle(NULL), NULL);
SendMessage(hWinEdit, EM_SETLIMITTEXT, (WPARAM)MAXTEXT, (LPARAM)0); // EDIT: Erhoehe 64K Std
#endif
if (!hWinEdit)
  MessageBox(hWnd, TEXT("Kann Editorfenster nicht laden!"), appTitle, MB_ICONWARNING);

/* Initialisiere Font */
lfScr.lfQuality = PROOF_QUALITY;
if (lfScr.lfFaceName[0] == 0)
  GetObject(GetStockObject(SYSTEM_FONT), sizeof lfScr, (PTSTR)&lfScr); // Falls INI leer,
hFont = CreateFontIndirect(&lfScr); // nutze Systemfont
SendMessage(hWinEdit, WM_SETFONT, (WPARAM)hFont, (LPARAM)TRUE);

/* Initialisiere Farben */
#ifdef BUILD_UC
#ifdef BUILD_RE
ZeroMemory(&cr, sizeof cr);
cr.cbSize = sizeof cr;
cr.dwMask = CFM_COLOR;
cr.crTextColor = fgClr;
SendMessage(hWinEdit, EM_SETBKGNDCOLOR, (WPARAM)0, (LPARAM)bgClr);
// Microsoft: SCF_ALL-Nachricht soll nach WM_SETFONT erfolgen:
SendMessage(hWinEdit, EM_SETCHARFORMAT, (WPARAM)(UINT)SCF_ALL, (LPARAM)(CHARFORMAT*)&cr);
#endif
logbrush.lbHatch = 0;
logbrush.lbStyle = BS_SOLID;
logbrush.lbColor = bgClr;
hBrush = CreateBrushIndirect(&logbrush);
initColors(&choosecolor, customcolors, hWinEdit, fgClr); // Fuellt choosecolor und customcolors
#endif

/* Initialisiere teilweise Druckseite */
// Ein Teil der Druckseiteninitialisierung wird bereits hier statt erst in menuePageSetup()
// vorgenommen, damit auch ohne vorherigen Aufruf von menuePageSetup() andere Funktionen wie
// z.B. menuPrint() die Randbreite kennen bzw. eine solche ueberhaupt erst eingestellt ist.
// Die Variante, PageSetupDlg ohne Dialogbox als Initialisierung von DEVMODE/NAMES aufzurufen
// wird hier nicht verwendet, um keine evt. Wartezeit auf nicht vorhandene Drucker zu erzeugen.
ZeroMemory(&psd, sizeof(psd));
psd.lStructSize = sizeof(psd);
psd.hwndOwner = hWnd;
psd.Flags = PSD_INHUNDREDTHSOFMILLIMETERS | PSD_MARGINS;
psd.rtMargin.top = psd.rtMargin.left = psd.rtMargin.right = psd.rtMargin.bottom = 2500; // 25mm

/* Baue Child-Fenster Toolbar */
#ifdef BUILD_TB
hWinTool = CreateWindowEx(0, TOOLBARCLASSNAME, NULL, WS_CHILD, // Kein WS_VISIBLE, da per INI
  // Toolbar beim Start unsichtbar sein kann, Schaltung muss extra erfolgen
  0, 0, 0, 0, hWnd,(HMENU)IDW_WIN_TOOL, GetModuleHandle(NULL), NULL);
if (!hWinTool)
  MessageBox(hWnd, TEXT("Kann Werkzeugleiste nicht laden!"), appTitle, MB_ICONWARNING);
SendMessage(hWinTool, TB_BUTTONSTRUCTSIZE, (WPARAM)sizeof(TBBUTTON), 0); // MS Kompatibilitaet
tbab.hInst = HINST_COMMCTRL;
tbab.nID = IDB_STD_SMALL_COLOR;  // Alternativ: IDB_STD_LARGE_COLOR
SendMessage(hWinTool, TB_ADDBITMAP, 0, (LPARAM)&tbab); // Fuege Buttons hinzu
```

```
  SendMessage(hWinTool, TB_ADDBUTTONS, sizeof tbb / sizeof tbb[0],(LPARAM)&tbb);
  if (showWinTool)
    ShowWindow(hWinTool, SW_SHOW);
  #endif

  /* Baue Child-Fenster Statusleiste */
  hWinStat = CreateWindowEx(0, STATUSCLASSNAME, NULL, WS_CHILD | SBARS_SIZEGRIP,
    // Kein WS_VISIBLE, da per INI Statusleiste beim Start unsichtbar sein kann, Schaltung
    0, 0, 0, 0, hWnd, (HMENU)IDW_WIN_STAT, GetModuleHandle(NULL), NULL); // muss extra erfolgen
  if (!hWinStat)
    MessageBox(hWnd, TEXT("Kann Statusleiste nicht laden!"), appTitle, MB_ICONWARNING);
  SendMessage(hWinStat, SB_SETPARTS, sizeof statwidths/sizeof statwidths[0], (LPARAM)statwidths);
  // SendMessage(hWinStat, SB_SETTEXT, 0, (LPARAM)TEXT("Bereit"));
  if (showWinStat)
    ShowWindow(hWinStat, SW_SHOW);

  /* Melde Nachrichten fuer Suchen/Ersetzen-Dialog beim System an */
  #ifdef BUILD_FR
  fr_msg = RegisterWindowMessage(FINDMSGSTRING);
  #endif

  /* Lese Namen zu ladender Texte von Kommandozeile oder von Shell-Drop auf ed.exe ein */
  ut_getArg(GetCommandLine(), NULL); // Achtung: ut_getArg() mit (.., NULL) zu initialisieren!
  ut_getArg(NULL, s); // Ueberspringe "ed.exe"
  ut_getArg(NULL, s); // Lese Dateinamen
  if (*s)
    fileOpen(hWnd, s, 0);

  return 0;
  } // WM_CREATE

 case WM_SIZE:
  {
  RECT rcEdit, rcStat;
  #ifdef BUILD_TB
  RECT rcTool;
  #endif
  int iEditHeight, iStatHeight = 0, iToolHeight = 0;
  #ifdef BUILD_TB
  if (showWinTool) {
    SendMessage(hWinTool, TB_AUTOSIZE, 0, 0); GetWindowRect(hWinTool, &rcTool);
    iToolHeight = rcTool.bottom - rcTool.top;
    }
    #endif
  if (showWinStat) {
    SendMessage(hWinStat, WM_SIZE, 0, 0); GetWindowRect(hWinStat, &rcStat); iStatHeight =
      rcStat.bottom - rcStat.top; }
  GetClientRect(hWnd, &rcEdit); iEditHeight = rcEdit.bottom - iToolHeight - iStatHeight;
  SetWindowPos(hWinEdit, NULL, 0, iToolHeight, rcEdit.right, iEditHeight, SWP_NOZORDER);
  // Wie stellt man das Padding des EDIT-Control ein? An sich klingt SendMessage(hWinEdit,
  // EM_SETMARGINS... gut: die Einstellung wirkt permanent und muss daher nicht in WM_SIZE
  // untergebracht werden; geaendert wird aber nur der li. und re. Rand, nicht der ob. und unt.
  // Setzt man stattdessen EC_USEFONTINFO, werden Raender sogar verkleinert statt vergroessert!
  // Relativer Rand berechnet aus FontInfo:
  // SendMessage(hWinEdit, EM_SETMARGINS, (WPARAM)(EC_USEFONTINFO), 0);
  // Absoluter Rand: fkt. zwar, aber geht nicht auf variable Schriftgroesse ein: SendMessage
  // (hWinEdit, EM_SETMARGINS, (WPARAM)(EC_LEFTMARGIN | EC_RIGHTMARGIN), MAKELPARAM(30,30));
  // Daher Weg ueber EM_SETRECT gegangen, dessen Einstellung aber immer nur bis zum naechsten
  // WM_SIZE haelt; daher muss dieser Ansatz in WM_SIZE selbst untergebracht werden.
  #ifndef BUILD_RE
```

```
  // Unter RICHEDIT verkleinert sich Fensterinhalt bei jedem Sizing immer mehr, daher deaktiviert
  if (padding > 0) {
    SendMessage(hWinEdit, EM_GETRECT, 0, (LPARAM)(LPRECT)&rcEdit);
    rcEdit.left+=padding; rcEdit.top+=padding; rcEdit.right-=padding; rcEdit.bottom-=padding;
    SendMessage(hWinEdit, EM_SETRECT, 0, (LPARAM)(LPRECT)&rcEdit);
    }
  #endif
  return 0;
  }

#ifdef BUILD_EM
case WM_INITMENUPOPUP:
  switch (lParam) {
  case 0: /* Menue Datei */
    {
    /* Aktiviere Menueeintraege Neu, Speichern, Drucken, falls moeglich */
    // Restriktiver als die meisten Editoren:
    // - Speichern/Drucken nur bei nichtleeren Dateien gestattet
    // - Speichern nur bei readwrite gestattet
    // - Neue Datei nur moeglich, falls nichtleer oder benannte Datei aktuell geladen
    int textLen = GetWindowTextLength(hWinEdit);
    EnableMenuItem((HMENU)wParam, IDM_FILE_NEW, MESTAT(textLen || *filePath));
    EnableMenuItem((HMENU)wParam, IDM_FILE_SAVE, MESTAT(dirty && textLen));
    EnableMenuItem((HMENU)wParam, IDM_FILE_SAVEAS, MESTAT(textLen));
    EnableMenuItem((HMENU)wParam, IDM_FILE_PRINT, MESTAT(textLen));
    return 0;
    }
  case 1: /* Menue Bearbeiten */
    {
    /* Aktiviere Menueeintraege Rueckgaengig und Einfuegen, falls moeglich */
    int sel0, selz, enable; // READONLY verweigert Tippen/Einfuegen, nicht aber REPLACESEL,
    EnableMenuItem((HMENU)wParam, IDM_EDIT_UNDO, // INSDATE, daher hier
      MESTAT(!readOnly && SendMessage(hWinEdit,EM_CANUNDO,0,0L)));
    EnableMenuItem((HMENU)wParam, IDM_EDIT_PASTE,
      MESTAT(!readOnly && IsClipboardFormatAvailable(CF_TEXT)));
    SendMessage(hWinEdit, EM_GETSEL, (WPARAM)&sel0, (LPARAM)&selz);
    /* Aktiviere Menueeintraege Kopieren, Ausschneiden und Loeschen, falls Text ausgewaehlt */
    enable = sel0 != selz;
    EnableMenuItem((HMENU)wParam, IDM_EDIT_CUT, MESTAT(enable && !readOnly));
    EnableMenuItem((HMENU)wParam, IDM_EDIT_COPY, MESTAT(enable));
    EnableMenuItem((HMENU)wParam, IDM_EDIT_CLEAR, MESTAT(enable && !readOnly));
    /* Aktiviere Eintraege Suchen, Ersetzen und Weitersuchen, falls Dialoge nicht schon aktiv */
    #ifdef BUILD_FR
    EnableMenuItem((HMENU)wParam, IDM_EDIT_FIND, MESTAT(fr_hDlg==NULL));
    EnableMenuItem((HMENU)wParam, IDM_EDIT_NEXT, MESTAT(fr_hDlg==NULL));
    EnableMenuItem((HMENU)wParam, IDM_EDIT_REPL, MESTAT(fr_hDlg==NULL && !readOnly));
    #endif
    EnableMenuItem((HMENU)wParam, IDM_EDIT_INSDATE, MESTAT(!readOnly));
    return 0;
    }
  case 2: /* Menue Optionen */
    /* Setze Status bestimmter Menueeintraege */
    EnableMenuItem((HMENU)wParam, IDM_OPTION_FGCOL, MESTAT(!readOnly)); // V1.1 Fix
    EnableMenuItem((HMENU)wParam, IDM_OPTION_BGCOL, MESTAT(!readOnly)); // V1.1 Fix
    CheckMenuItem((HMENU)wParam, IDM_OPTION_PRINTPGNO, printPgNo?MF_CHECKED:MF_UNCHECKED);
    #ifdef BUILD_TB
    CheckMenuItem((HMENU)wParam, IDM_OPTION_SHOWTOOL, showWinTool?MF_CHECKED:MF_UNCHECKED);
    #endif
    CheckMenuItem((HMENU)wParam, IDM_OPTION_SHOWSTAT, showWinStat?MF_CHECKED:MF_UNCHECKED);
    CheckMenuItem((HMENU)wParam, IDM_OPTION_SAVE, saveOptions?MF_CHECKED:MF_UNCHECKED);
```

```
    return 0;
  }
#endif

#ifdef BUILD_UC
#ifndef BUILD_RE
case WM_CTLCOLOREDIT:
  {
  SetTextColor((HDC)wParam, fgClr);
  SetBkColor((HDC)wParam, bgClr);
  return (LRESULT)hBrush;
  }
#endif
#endif

case WM_SETFOCUS:
  SetFocus(hWinEdit);
  #ifdef BUILD_FR
  if (fr_hDlg) // Benutzer von Suchdialog in WIN_EDIT gewechselt; informiere FindRoutinen,
    fr_setFocusLost(); // dass urspr. Selektion invalidisiert wurde
  #endif
  return 0;

case WM_DROPFILES:
  {
  HDROP hDrop = (HDROP)wParam;
  if (DragQueryFile(hDrop, 0, s, MAX_PATH) > 0)
    if (!dirty || fileSaveBeforeExit(hWnd) != IDCANCEL)
      fileOpen(hWnd, s, 0);
  DragFinish(hDrop);
  return 0;
  }

case WM_CLOSE:
  if (!dirty || fileSaveBeforeExit(hWnd) != IDCANCEL)
    DestroyWindow(hWnd);
  return 0;

case WM_QUERYENDSESSION:
  return !dirty || fileSaveBeforeExit(hWnd) != IDCANCEL;

case WM_DESTROY:
  /* Schreibe Ini-Datei */
  if (saveOptions) {
    if (makeIniDir) { // V1.1 New
      // Gewinne Verzeichnis aus iniPath ohne "\ed.ini"
      lstrcpyn(s,iniPath,lstrlen(iniPath)-sizeof("ed.ini")+1);
      if (!CreateDirectory(s,NULL))
        ut_messageBoxExt(hWnd, TEXT("Kann Verzeichnis '%s' nicht anlegen!"), appTitle,
          MB_ICONWARNING, s);
      }
    if (WritePrivateProfileString(TEXT("Ini"), TEXT("Version"), TEXT("1.0"), iniPath) == FALSE)
      ut_messageBoxExt(hWnd, TEXT("Kann Optionen nicht in '%s' schreiben!"), appTitle,
        MB_ICONWARNING, iniPath);
    else {
      GetWindowRect(hWnd, &rcWin);
      WritePrivateProfileStruct(TEXT("Options"), TEXT("Window"), &rcWin, sizeof rcWin, iniPath);
      WritePrivateProfileString(TEXT("Options"), TEXT("ScrFontName"), lfScr.lfFaceName, iniPath);
      WritePrivateProfileString(TEXT("Options"), TEXT("ScrFontSize"),
        ut_uLong2Str(lfScr.lfHeight,10), iniPath);
```

```
      WritePrivateProfileString(TEXT("Options"), TEXT("ScrFontWeight"),
        ut_uLong2Str(lfScr.lfWeight,10), iniPath);
      WritePrivateProfileString(TEXT("Options"), TEXT("PrtFontName"), lfPrt.lfFaceName, iniPath);
      WritePrivateProfileString(TEXT("Options"), TEXT("PrtFontSize"),
        ut_uLong2Str(lfPrt.lfHeight,10), iniPath);
      WritePrivateProfileString(TEXT("Options"), TEXT("PrtFontWeight"),
        ut_uLong2Str(lfPrt.lfWeight,10), iniPath);
      #ifdef BUILD_UC
      WritePrivateProfileString(TEXT("Options"), TEXT("ScrFontColor1"), ut_revRGBStr(fgClr1),
        iniPath);
      WritePrivateProfileString(TEXT("Options"), TEXT("ScrBgColor1"), ut_revRGBStr(bgClr1),
        iniPath);
      WritePrivateProfileString(TEXT("Options"), TEXT("ScrFontColor2"), ut_revRGBStr(fgClr2),
        iniPath);
      WritePrivateProfileString(TEXT("Options"), TEXT("ScrBgColor2"), ut_revRGBStr(bgClr2),
        iniPath);
      #endif
      WritePrivateProfileString(TEXT("Options"), TEXT("StatusBar"),
        showWinStat ? TEXT("1") : TEXT("0"), iniPath);
      #ifdef BUILD_TB
      WritePrivateProfileString(TEXT("Options"), TEXT("ToolBar"),
        showWinTool ? TEXT("1") : TEXT("0"), iniPath);
      #endif
      }
    }
  DeleteObject(hFont);
  #ifdef BUILD_UC
  DeleteObject(hBrush);
  #endif
  PostQuitMessage(0);
  return 0;

#ifdef BUILD_ST
case IDM_UNHIDE:
  if ((UINT)wParam == IDI_ICON && (UINT)lParam == WM_LBUTTONDBLCLK) {
    st_DelIcon(hWnd, IDI_ICON);
    ShowWindow(hWnd, SW_RESTORE); // Jetzt wieder sichtbar, aber noch nicht aktiv
    SetForegroundWindow(hWnd); // Jetzt aktiv
    }
  return 0;
#endif

case WM_COMMAND:
  switch (LOWORD(wParam)) { // EDIT-Control sendet Nachrichten an Elternfenster via WM_COMMAND

  case IDW_WIN_EDIT:
    switch(HIWORD(wParam)) {

    case EN_UPDATE:
      // Bei RichEdit erst ab V.2 automatisch an Parent gesendet; fuer V1 muesste (wie fuer
      // EN_CHANGE) es erst via EM_SETEVENTMASK angemeldet werden. Leider wird schon das initiale
      // Textladen als Aenderung interpretiert.
      if (!dirty) {
        dirty = 1;
        SendDlgItemMessage(hWnd, IDW_WIN_STAT, SB_SETTEXT, STINDI, (LPARAM)TEXT("*"));
        return 0;
        }
      break;

    case EN_ERRSPACE: case EN_MAXTEXT:
```

```
      MessageBox(hWnd, TEXT("Maximale Editiergröße überschritten!"), appTitle, MB_ICONWARNING);
      return 0;
    }
    break; // IDW_WIN_EDIT

  case IDM_FILE_NEW:
    if (!dirty || fileSaveBeforeExit(hWnd) != IDCANCEL)
      menuFileNew(hWnd);
    return 0;

  case IDM_FILE_OPEN:
    if (!dirty || fileSaveBeforeExit(hWnd) != IDCANCEL)
      menuFileOpen(hWnd);
    return 0;

  case IDM_FILE_SAVE:
    menuFileSave(hWnd);
    return 0;

  case IDM_FILE_SAVEAS:
    menuFileSaveAs(hWnd);
    return 0;

  case IDM_FILE_PAGESETUP:
    menuFilePageSetup(&psd);
    return 0;

  case IDM_FILE_PRINT:
    menuFilePrint(hWnd, &psd, &lfPrt);
    return 0;

  case IDM_FILE_EXIT: // Wird nur bei Datei|Beenden angesprungen
    PostMessage(hWnd, WM_CLOSE, 0, 0);
    return 0;

  #ifdef BUILD_EM

  case IDM_EDIT_UNDO:
    SendMessage(hWinEdit, WM_UNDO, 0, 0);
    return 0;

  case IDM_EDIT_CUT:
    SendMessage(hWinEdit, WM_CUT, 0, 0);
    return 0;

  case IDM_EDIT_COPY:
    SendMessage(hWinEdit, WM_COPY, 0, 0);
    return 0;

  case IDM_EDIT_PASTE:
    SendMessage(hWinEdit, WM_PASTE, 0, 0);
    return 0;

  case IDM_EDIT_CLEAR:
    SendMessage(hWinEdit, WM_CLEAR, 0, 0);
    return 0;

  #endif

  case IDM_EDIT_SELALL:
```

```
  SendMessage(hWinEdit, EM_SETSEL, 0, -1);
  return 0;

 case IDM_EDIT_INSDATE:
  if (GetDateFormat(LOCALE_USER_DEFAULT, DATE_LONGDATE, NULL, NULL, s, sizeof s/sizeof s[0]))
    SendMessage(hWinEdit, EM_REPLACESEL, TRUE, (LPARAM)s);
  return 0;

 #ifdef BUILD_FR
 case IDM_EDIT_FIND: case IDM_EDIT_REPL:
  fr_hDlg = fr_dlg(hWnd, LOWORD(wParam)==IDM_EDIT_REPL);
  return 0;

 case IDM_EDIT_NEXT: // F3
  if (fr_getFindStr()[0] == TEXT('\0'))
    fr_hDlg = fr_dlg(hWnd, 0);
  else
    if(!fr_find(hWinEdit))
      ut_messageBoxExt(hWnd, TEXT("Kein weiterer Suchtext '%s' gefunden."), appTitle, MB_OK,
        fr_getFindStr());
  return 0;
 #endif

 case IDM_OPTION_FONTPRT:
  if (!menuOptionsFont(hWinEdit, &lfPrt, CF_PRINTERFONTS))
    break;
  return 0;

 case IDM_OPTION_FONTSCR:
  if (!menuOptionsFont(hWinEdit, &lfScr, CF_SCREENFONTS))
  // Befuellt ueber ChooseFont() cf und logFont. Ueber cf.rgbColors koennte aus FontDialog
  // benutzergewaehlte Textfarbe aktiviert werden; der Dialog bietet aber nur 16 Grundfarben,
  // daher nicht genutzt.
    break; // Achtung! Falls i.O., muss Code hier durchlaufen!

 case IDM_OPTION_FONTDEC: case IDM_OPTION_FONTINC:
  {
  HFONT hFontNew;
  if (LOWORD(wParam) == IDM_OPTION_FONTDEC)
    lfScr.lfHeight = (8 * lfScr.lfHeight) / 10;
  else if (LOWORD(wParam) == IDM_OPTION_FONTINC)
    lfScr.lfHeight = (10 * lfScr.lfHeight) / 8;
  hFontNew = CreateFontIndirect(&lfScr);
  SendMessage(hWinEdit, WM_SETFONT, (WPARAM)hFontNew, (LPARAM)TRUE);
  DeleteObject(hFont);
  hFont = hFontNew;
  if (padding)
    SendMessage(hWnd, WM_SIZE, 0, 0);
  return 0;
  }

 case IDM_OPTION_PRINTPGNO:
  printPgNo = !printPgNo;
  return 0;

 #ifdef BUILD_UC

 case IMD_OPTION_COLORMODE:
  nightMode = !nightMode;
  if (nightMode) {
```

```
      bgClr = bgClr2; fgClr = fgClr2; }
    else {
      bgClr = bgClr1; fgClr = fgClr1; }
    DeleteObject(hBrush);
    logbrush.lbColor = bgClr;
    hBrush = CreateBrushIndirect(&logbrush);
    #ifdef BUILD_RE
    cr.crTextColor = fgClr;
    SendMessage(hWinEdit, EM_SETCHARFORMAT, (WPARAM)(UINT)SCF_ALL, (LPARAM)(CHARFORMAT*)&cr);
    SendMessage(hWinEdit, EM_SETBKGNDCOLOR, (WPARAM)0, (LPARAM)bgClr);
    #else
    InvalidateRect(hWinEdit, 0, TRUE);
    #endif
    return 0;

  case IDM_OPTION_FGCOL: // Funktioniert nicht ohne Behandlung von WM_CTLCOLOREDIT
    if (ChooseColor(&choosecolor)) {
      fgClr = choosecolor.rgbResult;
      if (nightMode)
        fgClr2 = fgClr;
      else
        fgClr1 = fgClr;
      #ifdef BUILD_RE
      cr.crTextColor = fgClr;
      SendMessage(hWinEdit, EM_SETCHARFORMAT, (WPARAM)(UINT)SCF_ALL, (LPARAM)(CHARFORMAT*)&cr);
      #else
      InvalidateRect(hWinEdit, 0, FALSE);
      #endif
      }
    return 0;

  case IDM_OPTION_BGCOL: // Funktioniert nicht ohne Behandlung von WM_CTLCOLOREDIT
    if (ChooseColor(&choosecolor)) {
      DeleteObject(hBrush);
      bgClr = choosecolor.rgbResult;
      logbrush.lbColor = bgClr;
      hBrush = CreateBrushIndirect(&logbrush);
      if (nightMode)
        bgClr2 = bgClr;
      else
        bgClr1 = bgClr;
      #ifdef BUILD_RE
      SendMessage(hWinEdit, EM_SETBKGNDCOLOR, (WPARAM)0, (LPARAM)bgClr);
      #else
      InvalidateRect(hWinEdit, 0, TRUE);
      #endif
      }
    return 0;

  #endif

  #ifdef BUILD_TB
  case IDM_OPTION_SHOWTOOL:
    showWinTool = !showWinTool;
    ShowWindow(hWinTool, showWinTool ? SW_SHOW : SW_HIDE);
    SendMessage(hWnd, WM_SIZE, 0, 0); // Ruft WM_SIZE auf
    return 0;
  #endif

  case IDM_OPTION_SHOWSTAT:
```

```
  showWinStat = !showWinStat;
  ShowWindow(hWinStat, showWinStat ? SW_SHOW : SW_HIDE);
  SendMessage(hWnd, WM_SIZE, 0, 0); // Ruft WM_SIZE auf
  return 0;

 case IDM_OPTION_SAVE:
  saveOptions =! saveOptions;
  return 0;

 case IDM_OPTION_SHOWFULL:
  {
  static BOOL showFull, saveShowWinTool, saveShowWinStat;
  static HMENU hMenu;
  LONG winStyle;
  showFull = !showFull;
  // Sowohl SetMenu() als auch ShowWindow(hWnd) rufen WM_SIZE auf (vorausgesetzt, die Aufrufe
  // erfolgen verzoegert, sonst reduziert Windows die 2 WM_SIZE-Nachrichten auf 1), so dass ein
  // SendMessage(hWnd, WM_SIZE, 0, 0) unnoetig ist
  if (showFull) {
    saveShowWinTool = showWinTool; saveShowWinStat = showWinStat; // Sichere alten Status von
    showWinTool = showWinStat = FALSE; // Status/Toolbar und setz neuen (benoetigt von WM_SIZE)
    #ifdef BUILD_TB
    ShowWindow(hWinTool, SW_HIDE);
    #endif
    ShowWindow(hWinStat, SW_HIDE);
    ShowScrollBar(hWinEdit, SB_VERT, FALSE);
    hMenu = GetMenu(hWnd);
    SetMenu(hWnd, NULL);
    ShowWindow(hWnd, SW_SHOWMAXIMIZED);
    winStyle = GetWindowLong(hWnd, GWL_STYLE) & ~WS_CAPTION;
    SetWindowLong(hWnd, GWL_STYLE, winStyle);
    SetWindowPos(hWnd, 0, 0, 0, 0, 0, SWP_FRAMECHANGED | SWP_NOMOVE | SWP_NOSIZE);
    }
  else {
    showWinTool = saveShowWinTool; showWinStat = saveShowWinStat;
    #ifdef BUILD_TB
    ShowWindow(hWinTool, showWinTool ? SW_SHOW : SW_HIDE);
    #endif
    ShowWindow(hWinStat, showWinStat ? SW_SHOW : SW_HIDE);
    ShowScrollBar(hWinEdit, SB_VERT, TRUE);
    SetMenu(hWnd, hMenu);
    ShowWindow(hWnd, SW_RESTORE);
    winStyle = GetWindowLong(hWnd, GWL_STYLE) | WS_CAPTION;
    SetWindowLong(hWnd, GWL_STYLE, winStyle);
    SetWindowPos(hWnd, 0, 0, 0, 0, 0, SWP_FRAMECHANGED | SWP_NOMOVE | SWP_NOSIZE);
    }
  return 0;
  }

 case IDM_HELP_VER:
  DialogBox(GetModuleHandle(NULL), MAKEINTRESOURCE(IDD_VER), hWnd, (DLGPROC)verDlgProc);
  return 0;

 case IDM_HELP_COMMANDS:
  MessageBox(hWnd,
  TEXT("Strg N\tNeue Datei\n")
  TEXT("Strg O\tDatei öffnen\n")
  TEXT("Strg S\tDatei schließen\n")
  TEXT("Strg P\tDatei drucken\n")
  TEXT("Alt F4\tBeenden\n")
```

```
    TEXT("\n")
    TEXT("Alt Rück\tRückgängig\n")
    TEXT("Strg Z\tRückgängig\n")
    TEXT("Strg X\tAusschneiden\n")
    TEXT("Sh Entf\tAusschneiden\n")
    TEXT("Strg C\tKopieren\n")
    TEXT("Strg Einf\tKopieren\n")
    TEXT("Strg V\tEinfügen\n")
    TEXT("Sh Einf\tEinfügen\n")
    TEXT("Entf\tLöschen\n")
    TEXT("Rück\tLöschen\n")
    TEXT("Strg F\tSuchen\n")
    TEXT("F3\tWeitersuchen\n")
    TEXT("Strg R\tErsetzen\n")
    TEXT("Strg A\tAlles markieren\n")
    TEXT("Strg D\tDatum einfügen\n")
    TEXT("\n")
    TEXT("F2\tFarbwechsel\n")
    TEXT("F11\tVollbild an/aus\n")
    TEXT("Strg +\tSchrift vergrößern\n")
    TEXT("Strg -\tSchrift verkleinern\n")
    TEXT("Esc\tFlüchten\n")
    TEXT("F1\tHilfe"),
    TEXT("Tastenkürzel"), MB_OK);
    return 0;

  #ifdef BUILD_ST
  case IDM_HIDE:
    if (fr_hDlg) { // Offener, aber inaktiver Suchdialog? Dann schliesse ihn.
      DestroyWindow(fr_hDlg);
      fr_hDlg = NULL;
      return 0;
      }
    st_AddIcon(hWnd, IDI_ICON, LoadIcon(GetModuleHandle(NULL), MAKEINTRESOURCE(IDI_ICON)),
      appTitle);
    ShowWindow(hWnd, SW_HIDE);
    return 0;
  #endif

  } // LOWORD(wParam)
  break; // WM_COMMAND

 #ifdef BUILD_FR
 default: // Hier treffen durch fr_dlg() erzeugte Nachrichten fuer fr_find*() ein;
  if (msg == fr_msg) { // lParam haelt FR struct vor, hier nicht mehr benutzt
    int frCode = fr_dispatch(hWinEdit);
    switch(frCode) {
      case -3: fr_hDlg = NULL; break;
      case -2: MessageBox(hWnd, TEXT("Kein weiterer Suchtext gefunden."), appTitle,MB_OK); break;
      case -1: break;
      default: ut_messageBoxExt(hWnd, TEXT("%d Ersetzungen."), appTitle, MB_OK, frCode);
      }
    return 0;
    }
  break;
 #endif

 } // msg

 return DefWindowProc(hWnd, msg, wParam, lParam);
```

```
 }

/** Function: Main **/
int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmdLn, int nCmdShow)
 {
 HWND hWinMain;
 WNDCLASSEX classMain;
 INITCOMMONCONTROLSEX iccx;
 MSG msg;
 #ifdef BUILD_AC
 HACCEL hAccel;
 #endif

 iccx.dwSize = sizeof(INITCOMMONCONTROLSEX);
 iccx.dwICC = ICC_LINK_CLASS;
 if (!InitCommonControlsEx(&iccx))
  MessageBox(NULL, TEXT("Aufruf COMCTL32.DLL V6 gescheitert!"), appTitle, MB_ICONINFORMATION);

 /* Erstelle Hauptfensterklasse und verbinde sie mit Callback-Funktion von Hauptfenster */
 ZeroMemory(&classMain, sizeof classMain);
 classMain.cbSize = sizeof classMain;
 classMain.lpfnWndProc = winProc;
 classMain.hInstance = hInst;
 classMain.hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_ICON));
 classMain.hCursor = LoadCursor(NULL, IDC_ARROW);
 // classMain.hbrBackground = (HBRUSH)(COLOR_WINDOW+1); hWinEdit/hWinTool/hWinStat zeichnen selbst
 classMain.lpszMenuName = MAKEINTRESOURCE(IDM_MENU);
 classMain.lpszClassName = appTitle;
 classMain.hIconSm =
  (HICON)LoadImage(hInst, MAKEINTRESOURCE(IDI_ICON), IMAGE_ICON, 16, 16, 0);
 if (!RegisterClassEx(&classMain)) {
  #ifdef UNICODE
  MessageBox(NULL, TEXT("Unicode-Programm, Windows NT benötigt!"), appTitle, MB_ICONSTOP);
  #else
  MessageBox(NULL, TEXT("Kann Fensterklasse nicht registrieren!"), appTitle, MB_ICONSTOP);
  #endif
  return 1;
  }

 /* Erstelle Hauptfenster */
 hWinMain = CreateWindowEx(WS_EX_ACCEPTFILES, appTitle, appTitle,
  WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,
  CW_USEDEFAULT, NULL, NULL, hInst, NULL);
 if (hWinMain == NULL) {
  MessageBox(NULL, TEXT("Kann Hauptfenster nicht laden!"), appTitle, MB_ICONSTOP);
  return 1;
  }
 ShowWindow(hWinMain, nCmdShow);
 UpdateWindow(hWinMain);

 #ifdef BUILD_AC
 if ((hAccel=LoadAccelerators(hInst, MAKEINTRESOURCE(IDA_ACCEL))) == NULL)
  MessageBox(NULL, TEXT("Keine Accelerators. Resourcen gebunden?"), appTitle, MB_ICONWARNING);
 #endif

 /* Empfange Nachrichten der Dispatcher-Schleife */
 while (GetMessage(&msg, NULL, 0, 0) > 0)
  #ifdef BUILD_FR
  if (fr_hDlg == NULL || !IsDialogMessage(fr_hDlg, &msg)) // Somit Tab und ENTER durch Dialog
  #endif // statt durch winProc behandelt
```

```
    #ifdef BUILD_AC
    if (!TranslateAccelerator(hWinMain, hAccel, &msg))
    #endif
      {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
      }

#ifdef BUILD_RE
if (richEditLib)
  FreeLibrary(richEditLib); // Stuerzte in WM_DESTROY immer ab; am Main-Ende aber problemlos
#endif

return (int)msg.wParam;
}
```

3.5.2 fr.c

```
/** Editor ED - Modul FR: Suchen und Ersetzen **/

#include "top.h"

#ifdef BUILD_FR

#include <windows.h>
#include "ut.h"
#include "res.h"

#ifdef BUILD_RE

#error Die Suche sollte besser mit RICHEDIT-eigener Funktion implementiert werden!

#else

#define MAX_STRING_SIZE 256

HWND fr_hDlg;
static FINDREPLACE fr_dlgStruc;
static TCHAR fr_findStr[MAX_STRING_SIZE], fr_replStr[MAX_STRING_SIZE];
static int fr_findCt, fr_replCt, fr_focusLost;

#ifdef BUILD_FR_PR

/** Function: Besetze Suchmuster in Suchen-Ersetzen-Dialogbox mit markiertem Text vor **/
static void fr_preset(HWND hWinEdit)
 {
 int selLen;
 TCHAR *sel;
 if ((sel=ut_getEditSel(hWinEdit, &selLen)) == NULL)
   return;
 if (selLen >= MAX_STRING_SIZE)
   selLen = MAX_STRING_SIZE - 1;
 lstrcpyn(fr_findStr, sel, selLen+1); // Im Unterschied zu strncpy kopiert
 GlobalFree(sel); // lstrcpyn max. selLen-1 Zeichen, um NUL mitanzuhaengen!
 }

#endif

/** Function: Rufe Suchen-Ersetzen-Dialogbox auf **/
HWND fr_dlg(HWND hWnd, BOOL replDlg)
 {
 // fr_dlg ist nur eine Huelse und kehrt sofort zum Aufrufer zurueck; werden die Buttons betaetigt,
 // werden fr_msg-Nachrichten erzeugt, die die eigentlichen Arbeitsfunktionen fr_find*() ansteuern.
 // Nachfolgende Aenderungen von Suchoptionen (matchCase etc.) in fr_dlgStruc werden ueber Dialog
 // sofort an Variable fr_dlgStruc gegeben, sodass Arbeitsfunktionen immer aktuellen Zustand haben.
 fr_findCt = fr_replCt = 0;
 #ifdef BUILD_FR_PR
 fr_preset(GetDlgItem(hWnd, IDW_WIN_EDIT)); // fr_FindText mit evt. Markierung vorbesetzen
 #endif
 ZeroMemory(&fr_dlgStruc, sizeof(FINDREPLACE));
 fr_dlgStruc.lStructSize = sizeof(FINDREPLACE);
 fr_dlgStruc.hwndOwner = hWnd; // Bestimmt, an welches Fenster Nachrichten gesendet werden
 fr_dlgStruc.Flags = FR_DOWN | FR_NOUPDOWN; // DOWN voreinstellen und fixieren
 fr_dlgStruc.lpstrFindWhat = fr_findStr;
 fr_dlgStruc.wFindWhatLen = MAX_STRING_SIZE;
```

```
  if (replDlg) {
    fr_dlgStruc.lpstrReplaceWith = fr_replStr; fr_dlgStruc.wReplaceWithLen  = MAX_STRING_SIZE;
    return ReplaceText(&fr_dlgStruc);
    }
  else
    return FindText(&fr_dlgStruc); // Dialog aufgerufen; Find/ReplaceText warten Benutzereingaben
    // in Suchfelder NICHT ab, sondern kehren sofort mit Rueckgabe von hFindDlg zurueck (moduslos).
  }

// fr_find() wird durch den "Suchen"-Knopf im Suchdialog oder im Suchen-Ersetzen-Dialog
// aufgerufen, fr_replace() durch den "Ersetzen"-Knopf im Suchen-Ersetzen-Dialog.
// Beide Routinen bilden ein Gespann:
// fr_find() sucht und markiert Text, fr_replace() ersetzt ihn. fr_repl() muß jedoch intern
// (d.h., ohne, daß Benutzer "Suchen" drückt) manchmal erst fr_find() aufrufen, z.B. beim
// ersten Aufruf des Dialogs, wenn noch kein zu ersetzender Text markiert ist.

/** Function: Suche Suchmuster **/
BOOL fr_find(HWND hWinEdit)
  {
  int sos, eos, start;
  TCHAR *buf, *found;
  // Dokument in Puffer kopieren und dort suchen, da es leider kein direktes Find fuer EDIT-Controls
  // gibt, wohl aber fuer RICHEDIT! Puffer muss auch jedesmal neu angelegt werden, da bei einem
  // moduslosem Dialog der Benutzer Text zwischenzeitlich aendern koennte.
  fr_focusLost = 0;
  if ((buf=ut_getEditText(hWinEdit,NULL)) == NULL)
    return FALSE;
  /* Suche Text im Dokument */
  SendMessage(hWinEdit, EM_GETSEL, (WPARAM)&sos, (LPARAM)&eos);
  start = (sos != eos) ? sos+1 : sos;
  found = ut_strstr(buf+start, fr_dlgStruc.lpstrFindWhat, fr_dlgStruc.Flags&FR_MATCHCASE,
    fr_dlgStruc.Flags&FR_WHOLEWORD);
  if (found)
    start = (int)(found - buf); // FIX 1.4: Typecast fuer WIN64
  GlobalFree(buf);
  if (found == NULL)
    return FALSE;
  /* Markiere gefundenen Text fuer Benutzer */
  SendMessage(hWinEdit, EM_SETSEL, start, start+lstrlen(fr_dlgStruc.lpstrFindWhat));
  SendMessage(hWinEdit, EM_SCROLLCARET, 0, 0); // Aktualisiere Fenster, so dass Caret sichtbar wird
  ++fr_findCt;
  return TRUE;
  }

/** Function: Ersetze naechstes Suchmuster **/
static BOOL fr_repl(HWND hWinEdit)
  {
  int sos, eos, selIsValid;
  TCHAR *sel;
  SendMessage(hWinEdit, EM_GETSEL, (WPARAM)&sos, (LPARAM)&eos);

  /* A - FIND muss erst laufen ... */
  if (sos == eos) // weil kein Text zur Ersetzung markiert ist oder
    return fr_find(hWinEdit);
  if (fr_focusLost) { // weil der Benutzer zwischenzeitlich im EDIT-Control war
    selIsValid = 0; //  und die Markierung nicht mehr stimmen muss
    if ((sel=ut_getEditSel(hWinEdit, NULL)) != NULL) {
      selIsValid = lstrcmp(sel,fr_dlgStruc.lpstrFindWhat) == 0;
      GlobalFree(sel); // Markierung stimmt nicht!
      }
```

```
    if (!selIsValid) {
      SendMessage(hWinEdit, EM_SETSEL, sos, sos);
      return fr_find(hWinEdit);
      }
    }

  /* B - REPL */
  SendMessage(hWinEdit, EM_REPLACESEL, TRUE, (LPARAM)fr_dlgStruc.lpstrReplaceWith);
  ++fr_replCt;

  /* C - FIND markiert (sichtbar für Benutzer) den nächsten zu ersetzenden String */
  return fr_find(hWinEdit);
  }

/** Function: Ersetze alle Suchmuster **/
static UINT fr_replAll(HWND hWinEdit)
  {
  fr_findCt = fr_replCt = 0;
  SendMessage(hWinEdit, EM_SETSEL, 0, 0);
  while (fr_repl(hWinEdit))
    ;
  return fr_replCt;
  }

/** Function: Informiere andere Module ueber Suchmuster **/
TCHAR *fr_getFindStr(void)
  {
  return fr_findStr;
  }

/** Function: Dispatcher **/
int fr_dispatch(HWND hWinEdit)
  {
  // Returns:
  // -3 FR_DIALOGTERM
  // -2 No more found
  // -1 Something found
  // 0..n Count of replacements
  if (fr_dlgStruc.Flags & FR_DIALOGTERM)
    return -3;
  else
    if (fr_dlgStruc.Flags & FR_FINDNEXT)
      return fr_find(hWinEdit) ? -1 : -2;
    else
      if (fr_dlgStruc.Flags & FR_REPLACE)
        return fr_repl(hWinEdit) ? -1 : -2;
      else
        if (fr_dlgStruc.Flags & FR_REPLACEALL)
          return fr_replAll(hWinEdit);
        else
          return -4; // Unknown
  }

/** Function: Informiere FR-Modul, dass Benutzer in das Hauptfenster gewechselt war **/
void fr_setFocusLost(void)
  {
  fr_focusLost = 1; // fr_repl() fragt spaeter fr_setFocusLost ab und fr_find setzt es zurueck
  }

#endif
```

```
#endif
```

3.5.3 st.c

```
/** Editor ED - Modul ST: System Tray Management **/

#include "top.h"

#ifdef BUILD_ST

#include <windows.h>
#include "etc.h"
#include "res.h"

/** Function: Fuege Anwendungssymbol zum Taskleistenstatusfenster hinzu **/
BOOL st_AddIcon(HWND hWnd, UINT uID, HICON hicon, const TCHAR *lpszTip)
 {
 BOOL ret;
 NOTIFYICONDATA tnid;

 tnid.cbSize = sizeof(NOTIFYICONDATA);
 tnid.hWnd = hWnd; // Handle des Fensters, welches Callback-Nachrichten erhalten soll
 tnid.uID = uID; // Symbol-Bezeichner
 tnid.uFlags = NIF_MESSAGE | NIF_ICON | NIF_TIP;
 tnid.uCallbackMessage = IDM_UNHIDE;
 tnid.hIcon = hicon; // Symbol-Handle
 if (lpszTip) // Tooltip-Text
  lstrcpyn(tnid.szTip, lpszTip, sizeof tnid.szTip); // lpszTip ist auf 64b beschraenkt
 else
  tnid.szTip[0] = TEXT('\0');
 ret = Shell_NotifyIcon(NIM_ADD, &tnid);
 if (hicon)
  DestroyIcon(hicon);
 return ret; // TRUE, falls erfolgreich
 }

/** Function: Entferne Anwendungssymbol von Taskleistenstatusfenster **/
BOOL st_DelIcon(HWND hWnd, UINT uID)
 {
 BOOL ret;
 NOTIFYICONDATA tnid;

 tnid.cbSize = sizeof(NOTIFYICONDATA);
 tnid.hWnd = hWnd; // Handle des Fensters, welches Symbol hinzufuegte
 tnid.uID = uID; // Symbol-Bezeichner
 ret = Shell_NotifyIcon(NIM_DELETE, &tnid);
 return ret; // TRUE, falls erfolgreich
 }

#endif
```

3.5.4 ut.c

```
/** Editor ED - Modul UT: Utensilien **/

#include "top.h"
#include <windows.h>
#include <tchar.h>
#include <stdarg.h>

#define SWAPLONG(l) (((l) >> 16) | ((l) & 0x00FF00) | (((l) & 0x0000FF) << 16))

#ifndef uintptr_t
#ifdef _WIN64
typedef unsigned __int64 uintptr_t;
#else
typedef unsigned int uintptr_t;
#endif
#endif

/** Function: Erweiterte MessageBox-Funktion **/
int ut_messageBoxExt(HWND hWnd, const TCHAR *msg, const TCHAR *title, unsigned type, ...)
 {
 va_list varg;
 TCHAR s[256];
 va_start(varg, type); // Muss auf letztes Fixarg zeigen
 wvsprintf(s, msg, varg);
 va_end(varg); // LCC-Warn. ignorieren
 s[sizeof s / sizeof(TCHAR) - 1] = TEXT('\0');
 return MessageBox(hWnd, s, title, type);
 }

/** Function: Kopiere Text aus EDIT-Control in Puffer **/
TCHAR *ut_getEditText(HWND hWinEdit, int *len)
 {
 int bufLen;
 TCHAR *buf;
 if (len)
   *len = 0;
 if ((bufLen=GetWindowTextLength(hWinEdit)) == 0)
   return NULL;
 if ((buf=(TCHAR*)GlobalAlloc(GMEM_FIXED, (bufLen+1)*sizeof(TCHAR))) == NULL)
   return NULL; // Leider muss erst _gesamter_ Text kopiert werden, um Markierung zu erhalten
 GetWindowText(hWinEdit, buf, bufLen+1); // Selbst wenn Text aus Datei mit SetWindowTextA() auf
 if (len) // ANSI fixiert eingelesen wurde, befuellt GetWindowText() unter UNICODE (GetWindowTextW)
   *len = bufLen; // Puffer mit Unicode-Text - daher funktionieren Suchroutinen!
 return buf;
 }

/** Function: Kopiere markierten Text aus EDIT-Control in Puffer **/
TCHAR *ut_getEditSel(HWND hWinEdit, int *len)
 {
 int sos, eos, selLen;
 TCHAR *buf, *sel;
 if (len)
   *len = 0;
 SendMessage(hWinEdit, EM_GETSEL, (WPARAM)&sos, (LPARAM)&eos);
 if ((selLen=eos-sos) == 0)
   return NULL;
 if ((buf=ut_getEditText(hWinEdit, NULL)) == NULL)
```

```
    return NULL;
  if ((sel=(TCHAR*)GlobalAlloc(GMEM_FIXED, (selLen+1)*sizeof(TCHAR))) == NULL) {
    GlobalFree(buf); return NULL; }
  lstrcpyn(sel, buf+sos, selLen+1); // Unterschied zu strncpy: lstrcpyn kopiert nur n Zeichen,
  GlobalFree(buf); // wenn n+1 angegeben, um Null mitanzuhaengen!
  if (len)
    *len = selLen;
  return sel;
  }

/** Function: Wandele Zahl in Zeichenkette **/
TCHAR *ut_uLong2Str(unsigned long l, unsigned int radix)
  {
  static TCHAR s[13]; // Muss 13 sein, da printf Zahlen nicht trunkieren kann (Strings schon)!
  s[0] = 0;
  if (radix == 16)
    wsprintf(s, TEXT("%06lX"), l); // _ultot(l,s,16) erlaubt leider keine Formatbreite
  else if (radix == 10)
    wsprintf(s, TEXT("%ld"), l);
  return s;
  }

#ifdef BUILD_UC

/** Function: Wandele BGR-Zahl in RGB-Zeichenkette **/
TCHAR *ut_revRGBStr(unsigned long l)
  {
  return ut_uLong2Str(SWAPLONG(l),16); // BGR -> RGB
  }

/** Function: Wandele RGB-Zeichenkette in BGR-Zahl **/
unsigned long ut_revRGBVal(TCHAR *s)
  {
  unsigned long l = _tcstoul(s, NULL, 16); // Es gibt kein lstrtoul() in Windows
  return SWAPLONG(l); // RGB -> BGR
  }

#endif

/** Function: Ermittele Kommandozeilenargumente **/
BOOL ut_getArg(TCHAR s[], TCHAR arg[])
  {
  static TCHAR *psave;
  TCHAR stop, *p;

  /* Initialisiere */
  if (!arg) {
    if (!s)
      return FALSE;
    psave = s;
    return FALSE;
    }
  if (!psave) // Von nun an MUSS psave ein gueltiger Zeiger sein!
    return FALSE;
  /* Fahre an letzter Suchposition fort */
  *arg = TEXT('\0');
  p = psave;
  /* Ueberspringe fuehrende Leerzeichen */
  while (*p == TEXT(' '))
    ++p;
```

```
  if (!*p) {
    psave = NULL; return FALSE; }
  /* Ermittele Endezeichen */
  if (*p == TEXT("")) {
    stop = TEXT(""); ++p; }
  else
    stop = TEXT(' ');
  /* Merke Wortbeginn */
  psave = p;
  /* Laufe bis Endezeichen */
  do ++p; while (*p && *p != stop);
  /* Kopiere Zeichenkette */
  lstrcpyn(arg, psave, (int)(p-psave+1)); // Anders als strncpy kopiert lstrcpyn nur n Zeichen,
  if (!*p) // wenn n+1 angegeben, um 0 mitanzuhaengen!
    psave = p;
  else
    psave = p + 1;
  return TRUE;
  }

/** Function: Ergibt TRUE, wenn path als Ordner existiert **/
BOOL ut_dirExists(const TCHAR path[])
  {
  DWORD h;
  if (!path || path[0] == '\0')
    return FALSE;
  h = GetFileAttributes(path);
  if (h == 0xFFFFFFFF)
    return FALSE;
  return (BOOL) (h & FILE_ATTRIBUTE_DIRECTORY);
  }

/** Function: Ergibt TRUE, wenn path als Datei existiert **/
BOOL ut_fileExists(const TCHAR path[])
  {
  DWORD h;
  if (!path || path[0] == '\0')
    return FALSE;
  h = GetFileAttributes(path);
  if (h == 0xFFFFFFFF)
    return FALSE;
  return (BOOL) !(h & FILE_ATTRIBUTE_DIRECTORY);
  /* Oder:
  WIN32_FIND_DATA data;
  HANDLE handle = FindFirstFile(dirName,&data);
  if (handle != INVALID_HANDLE_VALUE) {
    FindClose(handle);
    return true;
    }
  return false;
  */
  }

#ifdef BUILD_FR

/** Function: Suche Zeichenkette in Zeichenkette ohne Beachtung von Gross- und Kleinschreibung **/
static TCHAR *_ut_strstri(const TCHAR *str, const TCHAR *pat)
  {
  // Rev. History:
  // 02/03/94 Fred Cole: Original
```

```
 // 07/04/95 Bob Stout: ANSI-fy
 // 16/07/97 Greg Thayer: Optimized
 // 09/01/03 Bob Stout: Bug fix (lines 40-41) per Fred Bulback
 // 12/09/08 As Dala: Modified

 // CharLower() wandelt sowohl Zeichenketten als auch Zeichen um; da aber Funktion als
 // LPTSTR CharLower(LPTSTR lpsz) deklariert ist, muessen wir Funktionsergebnis fuer Zeichen in
 // TCHAR* umwandeln, damit Kompiler Ruhe gibt. Wenn wir nur CharLower-Ergebnisse vergleichen,
 // kann allerdings Umwandlung des Funktionsergebnisses entfallen. Ausserdem muessen wir
 // Suchzeichen in WORD umwandeln, damit keine neg. Vorzeichenerw. passiert ('ü' -> int -4).
 // Ignoriere LCC-Warnungen: CharLower gibt 32bit zurück, falls Parameter ein Zeichen war.
 TCHAR *s0, *s, p0, *p;

 /* Init */
 s0 = (TCHAR *) str;
 p0 = (TCHAR)(uintptr_t)CharLower((TCHAR*)(uintptr_t)(*pat));
 while (*s0) {

  /* pat[0] in str enthalten? */
  while ((TCHAR)(uintptr_t)CharLower((TCHAR*)(uintptr_t)*s0) != p0) // Ignoriere Warn.
   if (!*++s0)
    return NULL;

  s = s0;
  p = (TCHAR *) pat;

  /* Ja. pat[1..n] in str enthalten? */
  do {
   ++s;
   ++p;
   if (!*p) // Falls  Ende des Suchmusters erreicht, ist Fund bestaetigt
    return (TCHAR *) s0;
   } while (CharLower((TCHAR*)(uintptr_t)*s) == CharLower((TCHAR*)(uintptr_t)*p));

  s0++;
  }
 return NULL;
 }

/** Function: Suche Zeichenkette in Zeichenkette **/
TCHAR *ut_strstr(const TCHAR *str, const TCHAR *pat, BOOL matchCase, BOOL wholeWord)
 {
 TCHAR *h = (TCHAR*)str;
 UINT patlen = lstrlen(pat);
 if (str == NULL || pat == NULL)
  return NULL;
 while ((h = matchCase ? _tcsstr(h,pat) : _ut_strstri(h,pat)) != NULL) {
  if (!wholeWord || ((h==str || !IsCharAlpha(*(h-1))) && !IsCharAlpha(*(h+patlen))))
   break;
  ++h;
  }
 return h;
 }


#endif
```

3.5.5 top.h

```
/** Editor ED - Header TOP: Kompilatsteuerung **/

#ifndef __TOP_H__
#define __TOP_H__


/* Unicode oder ANSI */
#define UNICODE

/* Kompilierbereich */
#define BUILD_TB // Toolbar, funktioniert nicht unter LCC V3.8
#define BUILD_UC // Benutzerdefinierbare Farben
#define BUILD_AC // Shortcuts
#define BUILD_EM // Editmenue (statt nur Shortcuts und Kontextmenue)
#define BUILD_FR // Suchen/Ersetzen
#define BUILD_FR_PR // Belege Suchtext im Dialog mit evt. Textmarkierung vor
#define BUILD_ST // Verkleinerung in Systemtray
#define BUILD_RO // Graues Nurlesefenster fuer R/O-Datei
#define NO_BUILD_RE VER02 // Richedit statt MultilineEdit: VER01, VER02 oder nicht definiert

/* C-RTL Fix */
#ifdef UNICODE // UNICODE aktiviert unter Borland Breitfassung NUR der Windows-Funktionen.
#define _UNICODE // Borland aktiviert mit _UNICODE Breitfassung der eigenen RTL-Funktionen
#endif // ( _tcsrchr -> wcsrchr), setzt zusaetzlich UNICODE und damit Breitfassung der Windows-
// Funktionen (lstrlen -> lstrlenW). Unter MSC muessen ebf. beide Schalter gesetzt werden


#endif
```

3.5.6 etc.h

```
/** Editor ED - Header ETC: Compiler-Spezifica **/

#ifndef __ETC_H__
#define __ETC_H__


/* Borland C */
#ifdef __BORLANDC__

#define ICC_LINK_CLASS 0x00008000 // COMCTL V6 fuer BC V5.5.1. Quelle: MSVC COMMCTRL.H
#define MAX_LINKID_TEXT 48
#define L_MAX_URL_LENGTH (2048 + 32 + sizeof("://"))

typedef struct tagLITEM {
  UINT mask;
  int iLink;
  UINT state;
  UINT stateMask;
  WCHAR szID[MAX_LINKID_TEXT];
  WCHAR szUrl[L_MAX_URL_LENGTH];
  } LITEM, *PLITEM;
typedef struct tagNMLINK {
  NMHDR hdr;
  LITEM item;
  } NMLINK, *PNMLINK;

#endif

/* LCC */
#ifdef __LCC__

#include <shellapi.h> // HDROP etc.
#define CC_ANYCOLOR 0x00000100

#endif


#endif
```

3.5.7 res.h

```
/** Editor ED - Header RES: Ressourcen **/

#ifndef __RES_H__
#define __RES_H__


#ifndef IDC_STATIC
#define IDC_STATIC (-1)
#endif

/* Bezeichner */
#define IDM_MENU 100
#define IDW_WIN_EDIT 201
#define IDW_WIN_TOOL 202
#define IDW_WIN_STAT 203
#define IDI_ICON 300
#define IDA_ACCEL 400
#define IDC_VERLINK 500
#define IDD_VER 501

/* Programmnachrichten */
#define IDM_FILE_NEW 1101
#define IDM_FILE_OPEN 1102
#define IDM_FILE_SAVE 1103
#define IDM_FILE_SAVEAS 1104
#define IDM_FILE_PAGESETUP 1105
#define IDM_FILE_PRINT 1106
#define IDM_FILE_EXIT 1107
#define IDM_EDIT_UNDO 1201
#define IDM_EDIT_CUT 1202
#define IDM_EDIT_COPY 1203
#define IDM_EDIT_PASTE 1204
#define IDM_EDIT_CLEAR 1205
#define IDM_EDIT_SELALL 1206
#define IDM_EDIT_INSDATE 1207
#define IDM_EDIT_FIND 1208
#define IDM_EDIT_NEXT 1209
#define IDM_EDIT_REPL 1210
#define IDM_OPTION_FONTSCR 1301
#define IDM_OPTION_FONTPRT 1302
#define IDM_OPTION_PRINTPGNO 1303
#define IDM_OPTION_FGCOL 1304
#define IDM_OPTION_BGCOL 1305
#define IDM_OPTION_SHOWTOOL 1306
#define IDM_OPTION_SHOWSTAT 1307
#define IDM_OPTION_SHOWFULL 1308
#define IDM_OPTION_SAVE 1309
#define IDM_OPTION_FONTINC 1310
#define IDM_OPTION_FONTDEC 1311
#define IMD_OPTION_COLORMODE 1312
#define IDM_HELP_COMMANDS 1401
#define IDM_HELP_VER 1402
#define IDM_HIDE 1501
#define IDM_UNHIDE 1502


#endif
```

3.5.8 fr.h

```
/** Editor ED - Header FR: Suchen und Ersetzen **/

#ifndef __FR_H__
#define __FR_H__


extern HWND fr_hDlg;

HWND fr_dlg(HWND hWnd, BOOL replDlg);
BOOL fr_find(HWND hWinEdit);
TCHAR *fr_getFindStr(void);
int fr_dispatch(HWND hWinEdit);
void fr_setFocusLost(void);


#endif
```

3.5.9 st.h

```
/** Editor ED - Header ST: System Tray Management **/

#ifndef __ST_H__
#define __ST_H__


BOOL st_AddIcon(HWND hWnd, UINT uID, HICON hicon, const TCHAR *lpszTip);
BOOL st_DelIcon(HWND hWnd, UINT uID);


#endif
```

3.5.10 ut.h

```
/** Editor ED - Header UT: Utensilien **/

#ifndef __UT_H__
#define __UT_H__


int ut_messageBoxExt(HWND hWnd, const TCHAR *msg, const TCHAR *title, unsigned type, ...);
TCHAR *ut_getEditText(HWND hWinEdit, int *len);
TCHAR *ut_getEditSel(HWND hWinEdit, int *len);
TCHAR *ut_uLong2Str(unsigned long l, unsigned int radix);
unsigned long ut_revRGBVal(TCHAR *s);
TCHAR *ut_revRGBStr(unsigned long l);
BOOL ut_getArg(TCHAR s[], TCHAR arg[]);
BOOL ut_dirExists(const TCHAR path[]);
BOOL ut_fileExists(const TCHAR path[]);
TCHAR *ut_strstr(const TCHAR *txt, const TCHAR *pat, BOOL matchCase, BOOL wholeWord);


#endif
```

3.5.11 ed.rc

```
/** Editor ED - Ressourcen **/

#include "top.h"
#include <windows.h>
#include <winver.h> // Fuer LCC
#include "res.h"

/* Sprache */
LANGUAGE LANG_GERMAN, SUBLANG_NEUTRAL

/* Menue */
IDM_MENU MENU
 BEGIN
 POPUP "&Datei"
  BEGIN
  MENUITEM "&Neu\tStrg+N", IDM_FILE_NEW, GRAYED
  MENUITEM "Ö&ffnen ...\tStrg+O", IDM_FILE_OPEN
  MENUITEM "&Speichern\tStrg+S", IDM_FILE_SAVE //, GRAYED
  MENUITEM "Speichern &unter ...", IDM_FILE_SAVEAS //, GRAYED
  MENUITEM "Seite ein&richten ...", IDM_FILE_PAGESETUP
  MENUITEM "&Drucken ...\tStrg+P", IDM_FILE_PRINT //, GRAYED
  MENUITEM SEPARATOR
  MENUITEM "&Beenden", IDM_FILE_EXIT
  END
 #ifdef BUILD_EM
 POPUP "&Bearbeiten"
  BEGIN
  MENUITEM "&Rückgängig\tStrg+Z", IDM_EDIT_UNDO
  MENUITEM "&Ausschneiden\tStrg+X", IDM_EDIT_CUT
  MENUITEM "&Kopieren\tStrg+C", IDM_EDIT_COPY
  MENUITEM "&Einfügen\tStrg+V", IDM_EDIT_PASTE
  MENUITEM "&Löschen\tEntf", IDM_EDIT_CLEAR
  #ifdef BUILD_FR
  MENUITEM SEPARATOR
  MENUITEM "&Suchen ...\tCtrl+F", IDM_EDIT_FIND
  MENUITEM "&Weitersuchen\tF3", IDM_EDIT_NEXT
  MENUITEM "&Ersetzen ...\tCtrl+R", IDM_EDIT_REPL
  #endif
  MENUITEM SEPARATOR
  MENUITEM "Alles &markieren\tStrg+A", IDM_EDIT_SELALL
  MENUITEM "&Datum einfügen\tStrg+D", IDM_EDIT_INSDATE
  END
 #endif
 POPUP "&Optionen"
  BEGIN
  MENUITEM "&Schriftart ...", IDM_OPTION_FONTSCR
  #ifdef BUILD_UC
  MENUITEM "Schrift&farbe ...", IDM_OPTION_FGCOL
  MENUITEM "&Hintergrundfarbe ...", IDM_OPTION_BGCOL
  #endif
  MENUITEM SEPARATOR
  MENUITEM "&Druckerschriftart ...", IDM_OPTION_FONTPRT
  MENUITEM "Drucke Seiten&zahl", IDM_OPTION_PRINTPGNO, CHECKED
  MENUITEM SEPARATOR
  #ifdef BUILD_TB
  MENUITEM "&Toolbar", IDM_OPTION_SHOWTOOL
  #endif
```

```
    MENUITEM "Status&leiste", IDM_OPTION_SHOWSTAT
    MENUITEM SEPARATOR
    MENUITEM "&Vollbild\tF11", IDM_OPTION_SHOWFULL
    MENUITEM SEPARATOR
    MENUITEM "&Beim Beenden speichern", IDM_OPTION_SAVE
    END
  POPUP "&?"
    BEGIN
    MENUITEM "&Tastatur", IDM_HELP_COMMANDS
    MENUITEM "&Version", IDM_HELP_VER
    END
  END

/* Tastenkuerzel */
#ifdef BUILD_AC
IDA_ACCEL ACCELERATORS
  BEGIN
  #ifdef BUILD_ST
  VK_ESCAPE, IDM_HIDE, VIRTKEY
  #endif
  "^N", IDM_FILE_NEW, ASCII
  "^O", IDM_FILE_OPEN, ASCII
  "^S", IDM_FILE_SAVE, ASCII
  "^P", IDM_FILE_PRINT, ASCII
  VK_BACK, IDM_EDIT_UNDO, VIRTKEY, ALT // In EDIT-Control nicht implementiert, daher hier nachgeholt
  #ifdef BUILD_FR
  "^F", IDM_EDIT_FIND, ASCII
  VK_F3, IDM_EDIT_NEXT, VIRTKEY
  "^R", IDM_EDIT_REPL, ASCII
  #endif
  "^A", IDM_EDIT_SELALL, ASCII // In EDIT-Control nicht implementiert, daher hier nachgeholt
  "^D", IDM_EDIT_INSDATE, ASCII
  VK_F2, IMD_OPTION_COLORMODE, VIRTKEY
  VK_F11, IDM_OPTION_SHOWFULL, VIRTKEY
  VK_ADD, IDM_OPTION_FONTINC, VIRTKEY, CONTROL
  VK_SUBTRACT, IDM_OPTION_FONTDEC, VIRTKEY, CONTROL
  VK_F1, IDM_HELP_COMMANDS, VIRTKEY
  END
#endif

/* Dialoge */
IDD_VER DIALOGEX 60, 20, 190, 48
STYLE DS_MODALFRAME | DS_SETFONT | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Version"
#ifdef _WIN64
FONT 9, "Segoe UI"
#else
FONT 8, "MS Shell Dlg"
#endif
  BEGIN
  DEFPUSHBUTTON "&OK", IDOK, 150, 15, 27, 17
  ICON IDI_ICON, IDC_STATIC, 13, 14, 21, 20
  LTEXT "Editor in C und Win-API", IDC_STATIC, 50, 10, 100, 9
  LTEXT "Version 1.2.5.2", IDC_STATIC, 50, 20, 100, 9
  CONTROL "2008-2017 <a href=""http://www.asdala.de/algorithmik/ed/"">asdala.de</a>",
  IDC_VERLINK, "SysLink", WS_TABSTOP, 50, 30, 100, 9
  END

/* Bild */
IDI_ICON ICON "ed.ico"
```

```
/* Version */
VS_VERSION_INFO VERSIONINFO
  FILEVERSION 1,2,5,2
  PRODUCTVERSION 1,2,5,2
  FILEFLAGSMASK 0x3FL
  #ifdef DEBUG
  FILEFLAGS 0x1L
  #else
  FILEFLAGS 0x0L
  #endif
  FILEOS 0x4L
  FILETYPE 0x1L
  FILESUBTYPE 0x0L
  BEGIN
  BLOCK "StringFileInfo"
    BEGIN
    BLOCK "040704B0"
    BEGIN
    VALUE "CompanyName", "asdala.de\0"
    VALUE "FileDescription", "Editor in C und Win-API\0"
    VALUE "FileVersion", "1.2.5.2\0"
    VALUE "InternalName", "ED\0"
    VALUE "LegalCopyright", "2008-2017 asdala.de\0"
    VALUE "OriginalFilename", "ed.exe\0"
    VALUE "ProductName", "ED\0"
    VALUE "ProductVersion", "1.2\0"
    END
    END
  BLOCK "VarFileInfo"
    BEGIN
    VALUE "Translation", 0x0407, 0x04B0
    END
  END

/* Manifest */
#ifdef _WIN64
1 RT_MANIFEST ed64.exe.manifest
#else
1 RT_MANIFEST ed.exe.manifest
#endif
```

3.5.12 ed.exe.manifest

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">

<assemblyIdentity name="Asdala.Windows.Editor" version="1.2.5.2" type="win32"/>

<description>Editor in C und Win-API</description>

<dependency>
<dependentAssembly>
<assemblyIdentity name="Microsoft.Windows.Common-Controls" version="6.0.0.0" type="win32"
processorArchitecture="*" publicKeyToken="6595b64144ccf1df" language="*"/>
</dependentAssembly>
</dependency>

<trustInfo xmlns="urn:schemas-microsoft-com:asm.v2">
<security>
<requestedPrivileges>
<requestedExecutionLevel level="asInvoker" uiAccess="false"/>
</requestedPrivileges>
</security>
</trustInfo>

<compatibility xmlns="urn:schemas-microsoft-com:compatibility.v1">
<application>
<supportedOS Id="{e2011457-1546-43c5-a5fe-008deee3d3f0}"/>
<supportedOS Id="{35138b9a-5d96-4fbd-8e2d-a2440225f93a}"/>
<supportedOS Id="{4a2f28e3-53b9-4441-ba9c-d69d4a4a6e38}"/>
<supportedOS Id="{1f676c76-80e1-4239-95bb-83d0f6d0da78}"/>
<supportedOS Id="{8e0f7a12-bfb3-4fe8-b9a5-48fd50a15a9a}"/>
</application>
</compatibility>

<application xmlns="urn:schemas-microsoft-com:asm.v3">
<windowsSettings>
<dpiAware xmlns="http://schemas.microsoft.com/SMI/2005/WindowsSettings">true</dpiAware>
</windowsSettings>
</application>

</assembly>
```