Objekte 3 | Datendateien | C - Linux - Arduino - Raspberry

Inhaltsverzeichnis

Homepage

Homepage und Downloads: www.schmuckhexen.at/programms

Allgemeine Beschreibungen

Die allgemeinen Dokumentationen findet man unter c/1_Dokus oder im Internet:

Vorwort:	www.schmuckhexen.at/programs/c/clar_vorwort.pdf	c/1_Dokus/clar_vorwort.pdf
Projekt c/ einrichten. Die ersten Schritte:	www.schmuckhexen.at/programs/c/clar_start.pdf	c/1_Dokus/clar_start.pdf
Projekthilfe und Projektmanager:	www.schmuckhexen.at/programs/c/clar_chelp.pdf	c/1_Dokus/clar_chelp.pdf
Ein neues C Programm erstellen:	www.schmuckhexen.at/programs/c/clar_projekt.pdf	c/1_Dokus/clar_projekt.pdf
Basisobjekte ohne Terminal In/Ouput:	www.schmuckhexen.at/programs/c/clar_objekte1.pdf	c/1_Dokus/clar_objekte1.pdf
Terminalsteuerung Box-Objekte für In/Ouput:	www.schmuckhexen.at/programs/c/clar_objekte2.pdf	c/1_Dokus/clar_objekte2.pdf

Dokumentation: C-Programme und Bibliotheken

- ▷ Die ersten Infos zu den C-Programmen oder Bibliotheken findet man in der Hilfedatei '1_read.me' im jeweiligen Programmordner.
- ▷ Für aufwendige Programme gibt es Beschreibungen im Format *.odt oder *.pdf im Ordner name/bin/_name/
- ▷ Die Dokumentation des Programmcodes befindet sich in den C-Headern der Programme oder Bibliotheken.
- ▷ Hilfe zu den fertigen Programmen liefert immer die Startoption '-h'.

Einstiege:

Programm chelp	Menügesteuerter Zugriff auf alle Dokus
Projekt c/ Einstieg und Übersicht	c/1_read.me
Header/Dokus für Bibliotheksfunktionen	c/lib/1_read.me
Testprogramme für Bibliotheksfunktionen	c/libtest/1_read.me

Datendateien von Projekt /c

Für die Datenspeicherung verwendet Projekt c/ einheitlich Textdateien in lesbarem C-artigen Format.

Beispiele für Datendateien:

Konfigurationsdateien	Mit Variablen und Funktionen
Strukturarrays	Nur Daten ohne Variablen und Funktionen
Scriptdateien	Mit Variablen und externen Funktionen
Bashdateien	Bashbefehle mit Variablen
Freies Format	Die Bedeutung dieser Daten wird im jeweiligen Programm definiert

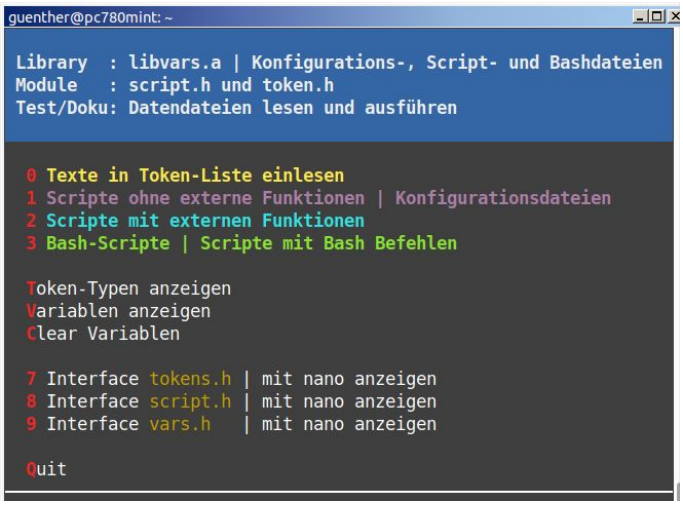
Datendateien werden im ersten Schritt mit der Funktion TokensReadFile() aus Objekt tTokens in funktionelle Einheiten (Token) zerlegt. Im zweiten Schritt wird die Liste der Token je nach gewünschter Syntax analysiert und verarbeitet.

Programm testscipt dokumentiert die Verarbeitung der Datendateien und ermöglicht es, die Datenfunktionen umfangreich zu testen.

Programm testscript anlegen:

```
chelp                 // Verwaltungsprogramm aufrufen
Programme/Libraries   // Programme compilieren
Projekt               // Projekt wählen
4 libtest             // Projektordner wählen
test_script/          // Datei wählen
Compiled              // Programm compilieren
Startlink             // Startlink anlegen
```

Das Programm testscript kann dann mit Befehl testscript gestartet werden.



Objekt tTokens

Beim Einlesen von Daten mit Funktion TokensReadFile() werden funktionelle Einheiten in einer Liste von Token gespeichert. Die Token erfassen einzelne Zeichen oder Zeichenfolgen des Textes.

Beispiel:

Der String "   Wert=25;" wir von TokensReadFile() in folgende Tokens zerlegt:

3 Blanks	Token Blank	{ .Typ=Blank, .s=" " } Feld s enthält die tatsächlichen Blanks
Wert	Token Name	{ .Typ=Name, .s="Wert" }
=	Token =	{ .Typ=, .s="=" }
25	Token Integer	{ .Typ=Integer, .s="25" }
;	Token ;	{ .Typ=;, .s=";" }

```
Token: tTokenTyp
 1: Blank
 2: \n
 3: Rem
 4: Name
 5: String
 6: Real
 7: Integer
 8: Hex
 9: Bin
10: Mode
11: (
12: )
13: {
14: }
15: [
16: ]
17: =
18: +
19: %
20: ,
21: ;
22: .
23: :
24: "
25: '
26: @
27: /
28: |
29: #
30: *
31: <
32: >
33: Ende
```

TokensReadFile()

In Bibliothek lib/vars Modul /c/lib/include/token.h findet man die Dokumentation zu TokensReadFile(). Die Funktion liest Textdateien und zerlegt sie in folgende Tokens:

Zeichen	Blank, usw.	Zeichen aus der nebenstehende Liste der Tokentypen
Kommentare	Rem	Zeilenkommentare **// ...** oder Blockkommentare **/* ... */**
Namen	Name	Zeichenfolgen für Bezeichner. $ ist im Bezeichner erlaubt.
String	String	Zeichenfolgen in **"..."** bei Option ParseStr=1
Direktiven	Mode	#Script Scriptmodus ein | ParseStr=1 | Strings speichern
		#Bash [ChkESC] Bashmodus ein | ParseStr=0 | Strings nicht speichern
		#Include "Dateiname" Der folgende Text wurde aus einer Datei eingefügt

Die Zerlegung in Token ist reversibel. Aus der Tokenliste kann wieder die Originaldatei rekonstruiert werden. Ausnahme: Blockkommentare **/* ... */** werden nicht als Token gespeichert.

Das Parsen von Strings **"..."** wird mit dem Flag ParseStr gesteuert. Das Flag kann beim Aufruf von TokensReadFile() gesetzt werden oder mit den Direktiven #Script oder #Bash im Text eingestellt werden:

ParseStr=1	Zeichen innerhalb von **"..."** werden als String-Token gespeichert. Dieser Modus kann durch die Direktive #Script gesetzt werden. Die Strings dürfen nur in einer Zeile liegen. Das ist keine Einschränkung, weil im zweiten Schritt durch die Stringverkettung einzelne Strings zusammengefasst werden: "StringA" "StringB" ergibt "StringAStringB"
ParseStr=0	Das Zeichen **"** wird als Token **"** gespeichert. Dieser Modus kann durch die Direktive #Bash gesetzt werden.

Datenverarbeitung

Im zweiten Schritt erfolgt die Analyse und Verarbeitung der Daten. Für Standardformate gibt es vordefinierte Funktionen:

readConfig()	Konfigurationsdatei lesen und die Werte in globalen Laufzeitvariablen speichern
ScriptRun()	Scriptdateien mit Variablen, externen Funktionen und Bashbefehlen laden und ausführen
DataArrayWR()	Strukturarrays speichern/lesen

Weitere "freie Datenformate" können sehr leicht implementiert werden. Funktionen zum Lesen/Schreiben können mit der Token-Liste und den Befehlen aus Objekt tTokens einfach programmiert werden:

Beispiel Datei test.timer: Siehe Doku devtest.

```
// Timer für Bewässerung
// Datei: test.timer
// Test: Ab 6 Uhr einschalten.

s 6:0:0, 23:0:0 // s Steuerblock | Zeiten täglich von - bis
x 1      // Befehl: ein
w 20:00  // Wartezeit
x 0      // Befehl: aus
w 40:00  // Wartezeit
l 0      // loop, Befehl: aus
```

Aus dieser Datendatei erzeugt TokensReadFile() nebenstehende Token-Liste.

```
[00]Rem      |// Timer für Bewässerung|
[01]\n
[02]Rem      |// Datei: test.timer|
[03]\n
[04]Rem      |// Test: Ab 6 Uhr einschalten. |
[05]\n
[06]\n
[07]Name     |s|
[08]Blank    | |
[09]Integer  |6|
[10]:
[11]Integer  |0|
[12]:
[13]Integer  |0|
[14],
[15]Blank    | |
[16]Integer  |23|
[17]:
[18]Integer  |0|
[19]:
[20]Integer  |0|
[21]Blank    | |
[22]Rem      |// Steuerblock|
[23]\n
[24]Name     |x|
[25]Blank    | |
[26]Integer  |1|
[27]Blank    |      |
[28]Rem      |// Befehl: ein|
[29]\n
[30]Name     |w|
[31]Blank    | |
[32]Integer  |20|
[33]:
[34]Integer  |00|
[35]Blank    |  |
[36]Rem      |// Wartezeit|
...
```

Zum Analysieren von Token-Listen findet man in Bibliothek lib Modul /c/lib/include/token.h die entsprechenden Hilfsfunktionen.

Mit Programm testscript kann man das Beispiel testen.
Befehl: 0 Texte in Token-Liste einlesen | 0 Text | test.timer

Der Timer wird in Programm gardenctl verwendet. Beschreibung in Programm devtest.

Beispiele für Datendateien

Konfigurationsdateien

Die Konfigurationsdateien *.conf werden von fast jedem Programm aus Projekt c/ verwendet. Das das Lesen/Speichern der Daten erfolgt mit Hilfe globaler Laufzeitvariablen aus dem Var-Objekt.

Eine Kurze Erklärung zu den globalen Laufzeitvariablen:

Alle Laufzeitvariablen besitzen zusätzlich zu ihrem Wert eine Grp- und SubGrpNummer. Grp steht für Gruppe.
Beim Lesen/Shreiben von Variablen werden die Filter GrpFilter- und SubGrpFilter verwendet.
Die Gruppen und Filter werden mit Funktion VarSetGrpFlt(Grp, SubGrp, GrpFilter, GrpSubFilter) gesetzt.

Beispiel:

Grp/Flt einstellen:	VarSetGrpFlt(100, 2, true, false); // Grp=100 , SubGrp=2, GrpFilter=ein, GrpSubFilter=aus
	Sichtbarkeit: Alle Variablen der Grp 100 sind unabhängig von ihrer SubGrp sichtbar.
Variable lesen:	VarGetInt("Test"); // Bezeichner "Test", Wert abfragen
	Variable "Test" ist sichtbar : Rückgabe Variablenwert
	Variable "Test" ist unsichtbar: Rückgabe bei Integer 0. Stringrückgabe "". Kein Fehler!
Variable anlegen:	VarSetInt("Test", 25); // Bezeichner "Test", Wert 25 setzen
	Variable "Test" ist sichtbar : Der Wert von "Test" wird auf 25 geändert
	Variable "Test" ist unsichtbar: Variable "Test" wird in Grp=100/SubGrp=2 mit Wert 25 neu angelegt

Die vollständige Beschreibung findet man in Bibliothek vars Modul c/lib/include/script.h.

Beispiel: Konfigurationsdatei für chelp. Zum Testen mit testscript findet man dort eine Kopie von chelp.conf.

Zum Lesen der Konfigurationsdateien wird die Funktion readConfig(const char *Pfad, uint16_t Group, uint16_t Debug); verwendet.

```
bool readConfig( const char *Pfad, uint16_t Group, uint16_t Debug);
  //
  // Konfigurationsdatei Pfad lesen und Variablen und Blöcke {..} in Var
  // speichern. Fehlermeldungen anzeigen.
  // Group: Startwert für Var Gruppe/Filter ist VarSetGrpFlt(Group, 0, true, false).
  // Debug: 0-2. 1: Konfiguration zeilenweise anzeigen.
  //
  // Kommentare: // ...
  //             Kommentare bleiben beim Speichern erhalten.
  //             Ausnahme: Blockkommentare /* ... */
  //
  // Wertzuweisungen: Name=Wert;
  //                  Der Wert wird unter "Name" in Var in der aktuellen
  //                  Grp/SubGrp von gespeichert.
  //
  // Stringverkettung: s = a + "..."  oder s = a "..."
  //
  // Blöcke: Block[]={ Wert1,  23,  -0.50  ,0b1110111, 0xFA, "Text" ... }
  //         Blöcke bestehen aus durch ',' getrennten Werten. Diese Werte werden
  //         als anonyme Vars in der aktuellen Grp/SubGrp gespeichert.
  //         In Blöcken gehen Kommentare verloren.
  //
  // Funktionen: Gruppen/Filter in der Konfiguration setzen:
  //             VarSetGrpFlt(Grp, SubGrp, ChkGrp, ChkSubGrp);  // Parameteranzahl 1,2,3 oder 4; 1==true, 0==false
  //             VarSetGrpFlt("Script") ruft die Funktion ScriptSetGrps();
  //
  // Rückgabe: false, Konfiguration nicht oder unvollständig gelesen.
```

Gekürzte und kommentierte Konfigurationsdatei chelp.conf

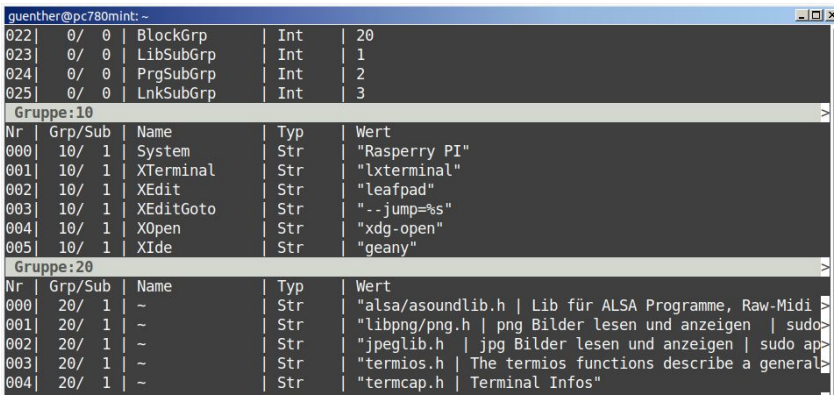
```
// ──────────────────────────────────────────────────
// 2025-03-15 chelp Konfiguration für für Pi und PC
//
// ──────────────────────────────────────────────────
// Die Allgemeine Einstellungen werden
// in der Programmgruppe 0/0 abgelegt.
VarSetGrpFlt(0,0);                                  | Die Gruppen auf Grp=0 und SubGrp=0 einstellen. Filter Default true/false

PathKeys  ="chelp.keys"; // Stichwortliste im gleichen       | Unter der Variblen PathKeys den String "chelp.keys" speichern
                         // Ordner wie chelp.conf
...

// Gruppen/Filter für verschiedene Systeme ──────────   | Gruppennummern speichern. Sie können im Script verwendet werden
//
ConfigGrp  = 10; // Konfigurationsgruppe                | Variable ConfigGrp auf 10 setzen
PXSubGrp   = 0;  // Subgruppe alle Systeme
PCSubGrp   = 1;  // Subgruppe für X Programme am PC
PISubGrp   = 2;  // Subgruppe für X Programme am Rasperry Pi

....

// ──────────────────────────────────────────────────
// X Einstellungen für PC
VarSetGrpFlt(ConfigGrp,PCSubGrp);                   | Gruppe/Filter für PC Einstellungen setzen

System       ="PC";            // Bezeichner
XTerminal    ="mate-terminal"; // X Terminal
XEdit        ="xed";           // X Editor
XEditGoto    ="+%s";           // X Viewer: Option Zeilennummer %s
XOpen        ="xdg-open";      // X Betrachter
XIde         ="geany";         // X IDE

VarSetGrpFlt(0,0); // Ende der Gruppe PC             | Gruppe/Filter zurücksetzen. Zugriff auf alle Variablen
// ──────────────────────────────────────────────────
```

Fortsetzung auf der nächsten Seite.

```
// ------------------------------------------------------------
// Gruppe/Filter für Blockvariablen                          | Gruppennummern für Blöcke. Sie können im Script verwendet werden
//
// Werte von Blöcken werden in anonymen Laufzeitvariablen
// gespeichert
//
BlockGrp  = 20;  // Gruppe Blockvariablen
LibSubGrp =  1;  // Subgruppe Library
PrgSubGrp =  2;  // Subgruppe Programme
LnkSubGrp =  3;  // Subgruppe Links

// ------------------------------------------------------------
// Projekt c/ | Bibliotheken überprüfen
// Zeilenaufbau: "Include | Beschreibung | Installation"
//
VarSetGrpFlt(BlockGrp,LibSubGrp);                            | Gruppe Blockvariablen / SubGruppe Library
Lib[]=                                                       | Es folgt eine Liste anonymer Variablen, also Variablen ohne Bezeichner
 {
 "alsa/asoundlib.h | Lib für ALSA Programme, Raw-Midi | sudo apt-get install libasound2-dev",
 "libpng/png.h | png Bilder lesen und anzeigen  | sudo apt-get install libpng-dev",
 "jpeglib.h  | jpg Bilder lesen und anzeigen | sudo apt-get install libjpeg-dev",
 "termios.h | The termios functions describe a general terminal interface",
 "termcap.h | Terminal Infos"};
VarSetGrpFlt(0,0);

...
```

Nach dem Einlesen von chelp.conf mit readConfig() stehen die Variablen als globale Laufzeitvariablen dem Programm zur Verfügung.

Auszug aus der Variablenliste:



Gruppen für Blockvariablen

PC Einstellungen

Anonyme Blockvariablen

Globale Laufzeitvariablen können unter einem beliebigen Dateipfad z.B. chelp.conf gespeichert werde. Dabei werden nur jene Laufzeitvariablen gesichert, die auch chelp.conf definiert sind.

```
bool updateConfig( const char *Pfad, uint16_t Group, uint16_t ZielModus, uint16_t Debug);
  // Konfigurationsdatei Pfad lesen und updaten. Fehlermeldung anzeigen.
  //
  // Pfad:      Konfigurationsdatei
  // Group:     Var Gruppe/Filter Startwert ist VarSetGrpFlt(Group, 0, true , false).
  // ZielModus: legt das Ausgabeziel fest.
  //            0: stdout
  //            1: tmp.cfg
  //            2: Pfad
  // Debug: Debugmodus 0..2. 0 keine Infos anzeigen
  //
  // Die Variablen und Blöcke {..} der Konfigurationsdatei werden
  // mit den Werten aus Var aktualisiert.
```

Strukturarrays

Datenstrukturen von Arrays können ebenfalls in Text-Dateien mit mit C-Struktur gespeichert oder gelesen werden.

Die Funktion DataArrayWR(DataDef) zum Speichern/Lesen von Arraystrukturen findet man in Bibliothek lib/var im Modul c/lib/include/data.h. Das zugehörige Test/Doku-Programm testrwstruct findet man unter c/libtest/test_rwstruct.

In Doku www.schmuckhexen.at/programs/c/clar_objekte1.pdf findet man eine kurze Einführung unter **Array speichern**.

Beispiel:

Datenbank dbebau.db für Programm dbebau. Die Datensätze werden durch nebenstehende Struktur tDb beschrieben.

```
//
// Array für Db-Objekt
//
// Deklaration eines Datensatzes vom Typ tDb:

typedef struct tDb       // Datensatz
{ time_t    Id;          // Id
  char     *Lager;       // Lager
  char     *Typ;         // Item Typ
  char     *Bezeichner;  // Bezeichner
  char     *Aufdruck;    // Aufdruck
  char     *Package;     // Gehaeuse
  char     *Funktion;    // Funktion
  int16_t   Anzahl;      // Stückzahl
  int16_t   Used;        // Stückzahl in Verwendung
  char     *Infos;       // Infos zum Bauteil
} tDb;
```

Im Testprogramm testscript findet man eine Kopie der Datenbankdatei dbebau.db.

Befehl: 0 Texte in Token-Liste einlesen | 0 Text | dbebau.db

Die Datendatei dbebau.db zur Struktur tDb:

```
// Demo: Datenbank für elektronische Bauteile
// Datenformat: Arraystruktur
// Datenspeicherung von Arraystrukturen mit C-Syntax
// Test/Doku: testrwstruct
//

Bauteile[]=
{{Id="20230127_094510",
  Lager="BOX4/41",
  Typ="Buchse",
  Bezeichner="USB Buchse A",
  Aufdruck=" ",
  Package="5",
  Funktion="USB-Buchse, printbar",
  Anzahl=4,
  Used=1,
  Infos=" ",
 },
 {Id="20230127_094553",
  Lager="W1",
  Typ="Z-Diode",
  Bezeichner="3,3V",
  Aufdruck=" ",
  Package="2",
  Funktion=" ",
  Anzahl=5,
  Used=0,
  Infos="500mW",
 },
 ...
}
```

Die Tokenliste.

Befehle: 1 TokensReadFile Tokenliste speichern
Tokenliste anzeigen

```
[00]Rem      |// Demo: Datenbank für elektronische Bauteile|
[01]\n
[02]Rem      |// Datenformat: Arraystruktur|
[03]\n
[04]Rem      |// Datenspeicherung von Arraystrukturen mit C-Syntax|
[05]\n
[06]Rem      |// Test/Doku: testrwstruct|
[07]\n
[08]Rem      |//|
[09]\n
[10]\n
[11]Name     |Bauteile|
[12][
[13]]
[14]=
[15]\n
[16]{
[17]{
[18]Name     |Id|
[19]=
[20]String   |20230127_094510|
[21],
[22]\n
[23]Blank    |  |
[24]Name     |Lager|
[25]=
[26]String   |BOX4/41|
[27],
[28]\n
[29]Blank    |  |
[30]Name     |Typ|
[31]=
[32]String   |Buchse|
[33],
[34]\n
[35]Blank    |  |
[36]Name     |Bezeichner|
[37]=
[38]String   |USB Buchse A|
[39],
[40]\n
[41]Blank    |  |
[42]Name     |Aufdruck|
[43]=
```

Die Beschreibung der Felder der Datendatei dbebau.db erfolgt mit der Datendefinition DbDef:

```
// Db write/read ------------------------------------------------------------
//
// Datendefinition für Array 'DevLst' zum Schreiben und Lesen.
// DatTyp ist der im C-Programm  definierte Datentyp.
// TokTyp ist der Datentyp aus 'tokens.h' für die Speicherung in der Datei
//
//typedef struct tDb tDb;

tDataDef DbDef=
{ .a=NULL,                  // Arraypointer wird für das Hauptarray gesetzt!
  //                        // NULL in den DatenDef's für alle ARRAY-Felder!
  .Bez="Bauteile",          // NULL oder Arraybezeichner fürs Datenfile
  .StructSize=sizeof(tDb),  // Byteanzahl des Arrayitems tStruct2
  .StructFeldAnz=0,         // 0,unbestimmt. Wird aus 'StructDef' berechnet
  .StructDef=               // Feldbeschreibungen für tStruct2
  { // Feldname_struct       | Typ im Prog   | Feld-Offset in struct              | Typ im File
    { .FeldBez="Id",         .DatTyp=TIME,    .Offset=offsetof(tDb, Id),         .TokTyp=String },
    { .FeldBez="Lager",      .DatTyp=STRING,  .Offset=offsetof(tDb, Lager),      .TokTyp=String },
    { .FeldBez="Typ",        .DatTyp=STRING,  .Offset=offsetof(tDb, Typ),        .TokTyp=String },
    { .FeldBez="Bezeichner", .DatTyp=STRING,  .Offset=offsetof(tDb, Bezeichner), .TokTyp=String },
    { .FeldBez="Aufdruck",   .DatTyp=STRING,  .Offset=offsetof(tDb, Aufdruck),   .TokTyp=String },
    { .FeldBez="Package",    .DatTyp=STRING,  .Offset=offsetof(tDb, Package),    .TokTyp=String },
    { .FeldBez="Funktion",   .DatTyp=STRING,  .Offset=offsetof(tDb, Funktion),   .TokTyp=String },
    { .FeldBez="Anzahl",     .DatTyp=INT16,   .Offset=offsetof(tDb, Anzahl),     .TokTyp=Integer},
    { .FeldBez="Used",       .DatTyp=INT16,   .Offset=offsetof(tDb, Used),       .TokTyp=Integer},
    { .FeldBez="Infos",      .DatTyp=STRING,  .Offset=offsetof(tDb, Infos),      .TokTyp=String },
    { .FeldBez=NULL },     // NULL: Ende der Definition
  },
};
```

Scriptdateien für saveit

Programm saveit dient zum Synchronisieren und Speichern von Daten. Die Funktionen des Programms können durch frei definierbare Menüs und Befehlsscripte definiert werden. Die Menüs werden mit den C-Strukturen aus Project /c definiert. Siehe: Menudateien für saveit.

Beispiel: Script mit externen Funktionen zum Sichern der einzelner Ordner des Projekt c/.

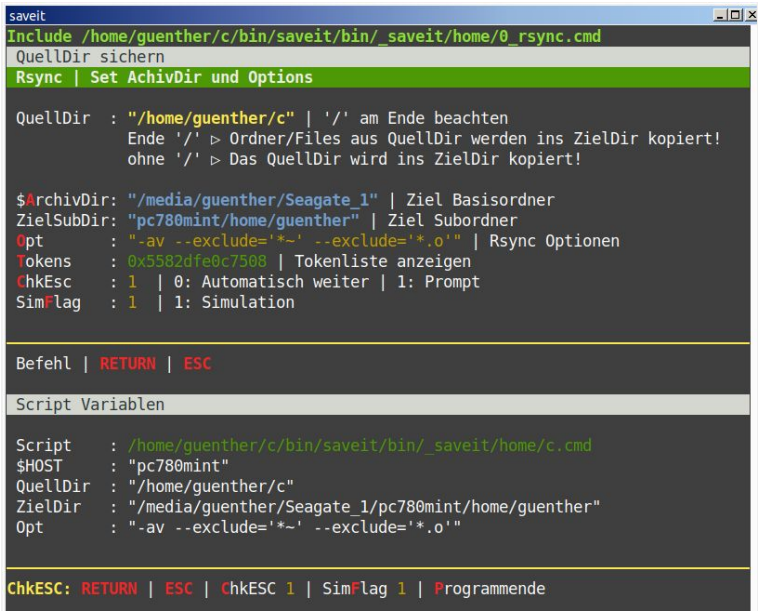
Script: c.cmd | Projektordner c sichern

```
// cmd | Rsync QuellDir auf ZielDir speichern
// Vars: $HOST, $HOME, $ArchivDir                | Die Variablen $HOST, $HOME, $ArchivDir werden vom Programm bereitgestellt
                                                 | Variablennamen mit $ können auch in Bashbefehlen verwendet werden

QuellDir = $HOME+"/c";                           | Ordner für Quelldateien festlegen
Opt= "-av --exclude='*~' --exclude='*.o'";       | Vorschlag für rsync Optionen: --exclude='*.o' keine Objektdateien kopieren

#Include "0_rsync.cmd"                           | Include-Script 0_rsync.cmd für die eigentlichen Sicherungsbefehle
                                                 | Dieses Script kann auch für andere Quellen verwendet werden
```

Script: 0_rsync.cmd

```
// cmd | Rsync QuellDir auf ZielDir speichern
// Vars: $HOST, $HOME, $ArchivDir                | Die Variablen $HOST, $HOME, $ArchivDir werden vom Programm bereitgestellt
//       QuellDir, Opt                           | Die Variablen QuellDir, Opt werden pm aufrufenden Script definiert

PrintTxt(" QuellDir sichern");                   | Textzeilen anzeigen. PrintTxt("Zeile1", "Zeile2", "Zeile3", ) kann beliebig viele
                                                 | Zeilen ausgeben. "Zeile1" wird als Überschrift formatiert

ZielSubDir  = $HOST+$HOME;                       | Subordner für das Ziel zusammenstellen
RsyncSetDlg("");                                 | Dialog zum Einstellen/Ändern der rsync Optionen: z.B- das $ArchivDir
ZielDir  = $ArchivDir+"/"+ZielSubDir;            | Zielordner zusammenstellen

PrintInfo("$HOST","QuellDir","ZielDir","Opt");   | PrintInfo zeigt den Scriptnamen und die gewählten Variablen an.
ChkESC();                                        | ChkESC() bietet eine Abbruchsmöglichkeit und Optionen für den weiteren Ablauf an.

ClrScr();                                        | Bildschirm löschen
Rsync(Opt, QuellDir, ZielDir);                   | Den externen Befehl Script_Rsync() aufrufen
ChkESC();
```

Die Ausführung:



Includedatei wurde eingefügt
PrintTxt(" QuellDir sichern");
RsyncSetDlg("");

ArchivDir suchen und wählen

Opt Optionendialog anzeigen/ändern

ChkESC 1: duchführen | 0: überspringen
SimFlag 1: Befehl nur anzeigen

PrintInfo("$HOST","QuellDir","ZielDir","Opt");

Die aktuellen Werte der Variablen anzeigen

ChkEsc() Abbruch oder den weiteren Ablauf wählen

ClrScr() Bildschirm wurde gelöscht
Script_Rsync() ausführen

Script_Rsync legt erforderliche Ordner an

Befehl anzeigen

SimFlag=1 Befehl nicht durchführen

Simulation: Befehle werden nur angezeigt

Beispiel: Script mit Bashbefehlen zum Erzeugen der Prüfsummen für das Archiv von Projekt c/.

Script: WWWShasumTar.cmd | Prüfsumme für c/ berechnen und im Text shasum.txt mit Anleitung speichern

```
// cmd | shasum.txt für c.tar.gz in www erzeugen
// Vars: $HOST, $HOME                                  | Die Variablen $HOST, $HOME werden vom Programm bereitgestellt

$ZielDir = "/var/www/html/programs/c";                 | Variablen festlegen. Bezeichner mit $ für die Namesersetzung im Bashbefehl
$Archiv  = "c.tar.gz";
$Befehl  = "shasum -a 256 c.tar.gz";
$Ziel    = "shasum.txt";                               | Dateiname für die Anleitung mit Prüfsumme

PrintTxt(                                              | Text zur Kontrolle anzeigen
 " Prüfsumme von $Archiv mit 'shasum a 256' berechnen", | Überschrift
 " ZielDir : " $ZielDir,
 " Archiv  : " $Archiv,
 " Befehl  : " $Befehl,
 " Ziel    : " $Ziel,
 ""
 );

ChDir($ZielDir);                                       | In den Zielordner wechseln
PrintInfo("$ZielDir","$Archiv", "$Befehl","$Ziel");    | Aktuelle Werte der Variablen anzeigen

#Bash ChkESC()                                         | 1. Bashbefehl
// Info nach Ziel schreiben                            | Dieser Kommentar wird bei der Ausführung angezeigt
echo                                                   | Bashbefehl echo. Anfang des Parmeterblocks. Layout sehr frei
 "Datum:      // Text Datum
  $(date)"    // aktuelles Datum abfragen              | $ wird nicht ersetzt, weil es keine Variable "$(" gibt
  > $Ziel;    // Zieldatei löschen, String schreiben   | $Ziel wird durch dem Wert der existierende Variablen $Ziel ersetzt
;                                                      | Blockende, Befehlsende

#Bash                                                  | 2. Bashbefehl
// Anleitung an das Ziel anhängen
echo 'Befehl: $Befehl' >> $Ziel;                       | Anleitung an die Datei $Ziel anhängen
;

#Bash ChkESC()                                         | 3. Bashbefehl
// Prüfsumme berechnen und an das Ziel anhängen
$Befehl >> $Ziel;                                      | Prüfsumme berechnen an Datei $Ziel anhängen
;
```

Die Ausführung:

```
saveit
 Prüfsumme von $Archiv mit 'shasum a 256' berechnen
 ZielDir : /var/www/html/programs/c
 Archiv  : c.tar.gz
 Befehl  : shasum -a 256 c.tar.gz
 Ziel    : shasum.txt

 ChDir
ChDir("/var/www/html/programs/c");

 Script Variablen

 Script    : /home/guenther/c/bin/saveit/bin/_saveit/c_sync/WWWShasumTar.cmd
 $ZielDir  : "/var/www/html/programs/c"
 $Archiv   : "c.tar.gz"
 $Befehl   : "shasum -a 256 c.tar.gz"
 $Ziel     : "shasum.txt"

Bashbefehl:0x55ddc4972b60 | Flags:1
// Infotext nach Ziel schreiben
Cmd: echo "Datum : $(date)" > shasum.txt

ChkESC: RETURN | ESC | ChkESC 1 | SimFlag 0 | Programmende

Bashbefehl:0x55ddc4972b60 | Flags:0
// Anleitung nach Ziel schreiben
Cmd: echo 'Befehl: shasum -a 256 c.tar.gz' >> shasum.txt

Bashbefehl:0x55ddc4972b60 | Flags:1
// Prüfsumme berechnen und nach Ziel schreiben
Cmd: shasum -a 256 c.tar.gz >> shasum.txt

ChkESC: RETURN | ESC | ChkESC 1 | SimFlag 0 | Programmende
```

PrintTxt(...)

ChDir($ZielDir)

PrintInfo(...)

Bashbefehl: Funktionspointer
Flags: 1: ChkESC() verwenden
Cmd: Befehlsstring ohne Kommentare

Menüdateien für saveit

In Bibliothek lib/iocon Modul c/lib/include/boxmenu.h findet man Funktionen und Strukturen für Menüs. Im Programm werden die Menüs mit Hilfe der Struktur tBoxMenuDef definiert. Dieselben Menüstrukturen können aber auch zur Laufzeit aus Menüdateien eingelesen werden.

Programm saveit verwendet automatisch die zum Rechner passende Menüs.

Menüdatei: pc780mint.menu | Menü für Rechner PC780

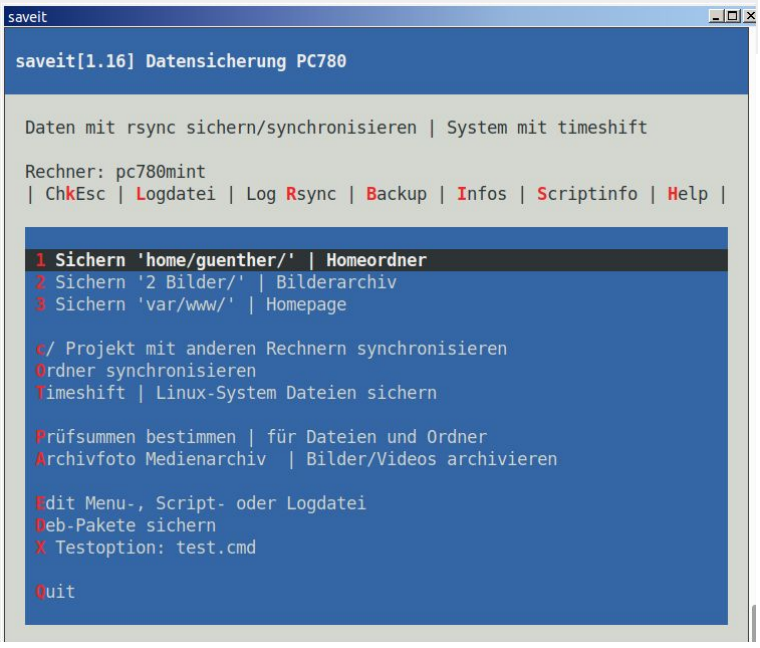
Der Eintrag Info wird im Programm durch weitere Funktionstasten ergänzt: | ChkEsc | Logdatei | usw.
Die Anzeige von Funktionstasten wird im Menüzeilentext mit \e markiert.
Die genaue Beschreibung der Menüstruktur findet man in In Bibliothek lib/iocon Modul c/lib/include/boxmenu.h.

```
// ========================================================
// saveit Menüdefinitionen für Rechner PC780
//
// Die Menüdefinition entspricht dem Typ tBoxMenuDef aus boxmenu.h!
// Alle Parameter sind Zahlen oder Strings. Der Feldselector '.' ist optional.
// '\et' Präfix für Funktionstasten in Rot.

Menu("MenuMain",        // Hautpmenu, obligatorisch!
  y=1, x=1, h=25, b=0,  // Position y, x, Höhe, Breite

  //FarbeTitel="\e[1m\e[43m",   // Farbe für Titel
  //FarbeZeile="\e[1m\e[43m"    // Farbe für Menüzeile
  Titel="Datensicherung PC780", // Menutitel

  Info="\n  Daten mit rsync sichern/synchronisieren | System mit timeshift\n",
  I=1,   // Startposition in Menuzeile 1
  Items=
  {{Typ="Leer",   Txt="", Ptr="" },
   {Typ="Menu",   Txt=" \et1 Sichern 'home/guenther/' | Homeordner",             Ptr="MenuHome",          Key="1" },
   {Typ="Menu",   Txt=" \et2 Sichern '2 Bilder/' | Bilderarchiv",                Ptr="MenuBilder",        Key="2" },
   {Typ="Script", Txt=" \et3 Sichern 'var/www/' | Homepage",                     Ptr="www/www.cmd",       Key="3" },
   {Typ="Leer",   Txt="", Ptr="" },
   {Typ="Menu",   Txt=" \etc/ Projekt mit anderen Rechnern synchronisieren",     Ptr="MenuC",             Key="c" },
   {Typ="Run",    Txt=" \etOrdner synchronisieren",                              Ptr="runSyncDirs",       Key="o" },
   {Typ="Run",    Txt=" \etTimeshift | Linux-System Dateien sichern",            Ptr="runTimeshift",      Key="t" },
   {Typ="Leer",   Txt="", Ptr="" },
   {Typ="Run",    Txt=" \etPrüfsummen bestimmen | für Dateien und Ordner",       Ptr="runChksum",         Key="p" },
   {Typ="Run",    Txt=" \etArchivfoto Medienarchiv  | Bilder/Videos archivieren", Ptr="runArchivfoto",     Key="a" },
   {Typ="Leer",   Txt="", Ptr="" },
   {Typ="Run",    Txt=" \etEdit Menu-, Script- oder Logdatei",                   Ptr="runEdit",           Key="e" },
   {Typ="Menu",   Txt=" \etDeb-Pakete sichern",                                  Ptr="MenuAPT",           Key="d" }
   {Typ="Script", Txt=" \etX Testoption: test.cmd",                              Ptr="test.cmd",          Key="x" },
   {Typ="Leer",   Txt="", Ptr="" },
   {Typ="Menu",   Txt=" \etQuit", Ptr="" }
  }
);


Menu("MenuHome",   // Homeordner sichern
  y=1, x=1, h=28, b=0,
  FarbeTitel="\e[1m\e[42m",  // Farbe Titel

  Titel="\n PC780: Ordner und Files aus HOME extern sichern\n", // Menutitel
  Info="\n  Ordner aus Home sichern\n",
  Items=
  { {Typ="Script", Txt=" '1 Dokus/'",      Ptr="home/1 Dokus.cmd" },
    {Typ="Script", Txt=" '3 Musik/'",      Ptr="home/3 Musik.cmd" },
     ...
  }
);

...
```

Die möglichen Menüeinträge:

```
typedef enum tBoxMenuTyp  // Typen für mögliche Menüeinträge
{ MenuNil=0,   // {} Ende der Menüeinträge! Für SizeOfMenu()!
  Run,         // Ptr ist Funktionspointer vom Typ tBoxMenuRun.
  Bash,        // Ptr ist Bash-Befehlstring für Systemaufruf.
  Script,      // Ptr ist ein Scriptname für Funktion ScriptRun().
  Menu,        // Ptr ist Menüdefinition vom Typ tBoxMenuDef.
  FTaste,      // kein Pt, Rückgabe globale Funktionstaste
  Leer,        // Leerzeile, kein Ptr
} tBoxMenuTyp;
```

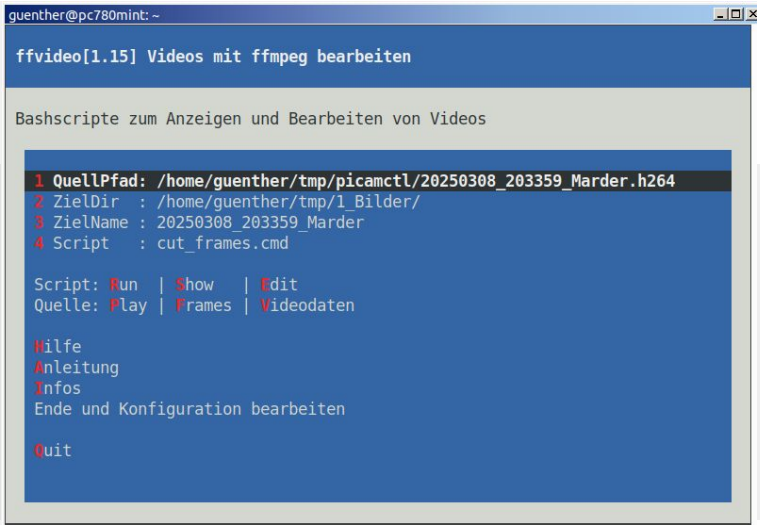
Das oben definierte MenuMain während der Ausführung:



Bashdateien für ffvideo

Programm ffvideo kann für die Bearbeitung von Videodateien mit Programm ffmpeg im Terminalfenster verwendet werden. Für verschiedenen Arbeitsschritte können entsprechende Befehlsdateien vorbereitet werden. Diese Befehlsdateien sind Scriptdateien mit Bashbefehlen. Sie werden mit ScriptRun(...) ausgeführt.

Anleitung: ffvideo.pdf



Beispiel: frames_nr.cmd

Die Variablen $Quelle und $QuelleSuffix werden vom Programm bereitgestellt.

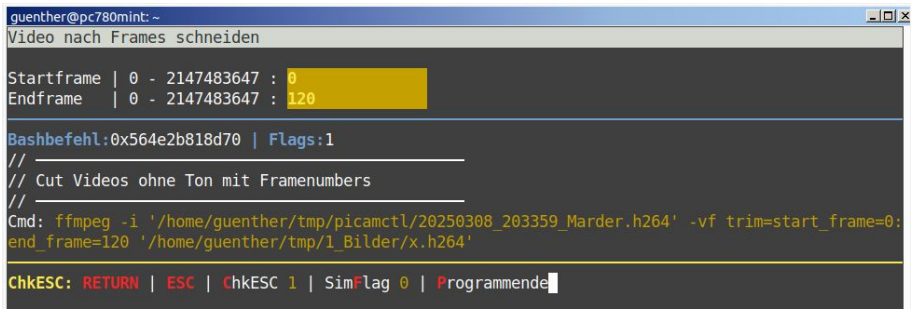
```
// ------------------------------------------------
// Vars: $Quelle und $QuelleSuffix

#Bash ChkESC()
// ------------------------------------------------
// Video abspielen und Framenummern einblenden
// ------------------------------------------------
ffplay
 '$Quelle$QuelleSuffix' // QuellPfad
 -vf
 "
 drawtext=fontfile=Arial.ttf:
 text='%{frame_num}':
 start_number=1: x=0: y=h-lh-4:
 fontcolor=black: fontsize=20:
 box=1: boxcolor=white: boxborderw=5
 "
 -hide_banner      // no Banner
;
```

Beispiel: frames_cut.cmd

Mit den oben angezeigtten Framenummern kann das Video dann geschnitten werden. Vars werden vom Programm bereitgestellt.

```
// frames_cut.cmd | Video nach Frames schneiden        | Das Script startet immmer Modus #Script
// Vars: $Quelle$, QuelleSuffix, $Ziel

PrintTxt("Video nach Frames schneiden","");           | Überschrift

ReadInt("$StartFrame","Startframe |","0");            | Framenummer mit ReadInt abfragen und unter $StartFrame speichern. Default: 0
ReadInt("$EndFrame"  ,"Endframe   |","0");

#Bash ChkESC()                                        | Bashbefehl mit ChkESC()
// ------------------------------------------------
// Cut Videos ohne Ton mit Framenummern               | Infotext anzeigen
// ------------------------------------------------
ffmpeg
 -i '$Quelle$QuelleSuffix'
 -vf trim=start_frame=$StartFrame:end_frame=$EndFrame
'$Ziel$QuelleSuffix'
;
```

Ausführung:



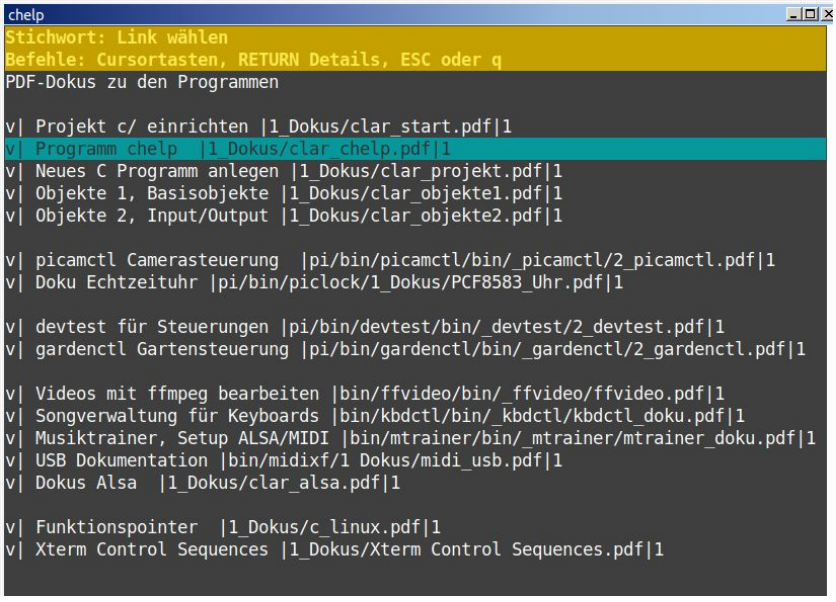
Stichwortliste für chelp

Mit der Stichwortliste aus Programm chelp können Texte angezeigt, Infos mit grep gesucht und Programme gestartet werden. Die Stichwortliste verwendet folgendes frei definiertes Datenformat:

Stichwortliste chelp.keys:

```
// ──────────────────────────────────────────────────────────────
// Stichwortliste für chelp | 2024-10-08
//
// Mit der Stichwortliste können Infos zum Projekt /c einfach
// angezeigt oder Programme gestartet werden.
//
// Alle Pfadangaben der Liste sind relativ zu c/ oder absolut.
// Die Angaben [..] sind optional.
// Kommentare //... am Ende eines Stichworteintrag werden angezeigt!
// Die Items vom Typ h,g und v erlauben Wildcards.
// Z.B  {v, "*.pdf" , "1" },  // alle PDF-Dateien zur Wahl
//
// Aufbau eines Eintrags der Stichwortliste:
//
// {"Stichwort-Info-Text",          Beschreibung des Eintrags
//  {t, "Text"                },    "Text" anzeigen
//  {h, "Datei", ["Suchstring"]},   Header aus Ordner include
//  {g, "Pfad" ,  "Suchstring" },   Dateien mit grep suchen
//  {v, "Pfad" , ["Nr"]        },   Viewer aufrufen. Zeilen "Nr"
//  {e, "Pfad" , ["Options"]   },   Execute Pfad mit Options
//  {m, "Datei", ["Nr"]        },   Manpage Datei. Seite Nr
//  ...
// };
// ...
// {"Stichwort-Info-Text",          Beschreibung des Eintrags
//  { ... },
//  ...
// };
// ──────────────────────────────────────────────────────────────

{"PDF-Dokus zu den Programmen",
 {v,"1_Dokus/clar_start.pdf","1"},    // Projekt c/ einrichten
 {v,"1_Dokus/clar_chelp.pdf","1"},    // Programm chelp                    | Link "Programm chelp" nach "1_Dokus/clar_chelp.pdf"
 {v,"1_Dokus/clar_projekt.pdf","1"},  // Neues C Programm anlegen
 {v,"1_Dokus/clar_objekte1.pdf","1"}, // Objekte 1, Basisobjekte
 {v,"1_Dokus/clar_objekte2.pdf","1"}, // Objekte 2, Input/Output
 {t,""},
 {v,"pi/bin/picamctl/bin/_picamctl/2_picamctl.pdf","1"}, // picamctl Camerasteuerung
 {v,"pi/bin/piclock/1_Dokus/PCF8583_Uhr.pdf","1"}, // Doku Echtzeituhr
 {t,""},
 {v,"pi/bin/devtest/bin/_devtest/2_devtest.pdf","1"}, // devtest für Steuerungen
 {v,"pi/bin/gardenctl/bin/_gardenctl/2_gardenctl.pdf","1"}, // gardenctl Gartensteuerung
 {t,""},
 {v,"bin/ffvideo/bin/_ffvideo/ffvideo.pdf","1"}, // Videos mit ffmpeg bearbeiten
 {v,"bin/kbdctl/bin/_kbdctl/kbdctl_doku.pdf","1"}, // Songverwaltung für Keyboards
 {v,"bin/mtrainer/bin/_mtrainer/mtrainer_doku.pdf","1"}, // Musiktrainer, Setup ALSA/MIDI
 {v,"bin/midixf/1 Dokus/midi_usb.pdf","1"}, // USB Dokumentation
 {v,"1_Dokus/clar_alsa.pdf","1"},     // Dokus Alsa
 {t,""},
 {v,"1_Dokus/c_linux.pdf","1"},       // Funktionspointer
 {v,"1_Dokus/Xterm Control Sequences.pdf","1"}, // Xterm Control Sequences
};

...
```

Ausführung der Stichwortliste in chelp:



GNU General Public License

```
/*
 *
 * Copyright 2022-2025 Günther Schardinger <v.schardinger@gmx.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
*/
```