Einführung in Tcl/Tk 8.6

Oliver Scholl

www.tcltk.info

2. Dezember 2023

Inhaltsverzeichnis

1 Vorwort 11

2 Über Tcl/Tk 13
2.1 Vorteile von Tcl/Tk . . . 13
2.2 Hilfreiche Internetseiten zu Tcl/Tk . . . 14
2.3 Bücher zu Tcl/Tk . . . 14

3 Beispiel-Programm 17

4 Installation 23
4.1 Installation unter Windows . . . 23
4.2 Installation unter Linux . . . 23

5 Tcl/Tk-Programme starten 25
5.1 Programme unter Windows starten . . . 25
5.1.1 Programme ohne GUI . . . 25
5.1.2 Programme mit GUI . . . 26
5.2 Programme unter Linux starten . . . 26
5.2.1 Programme ohne GUI . . . 26
5.2.2 Programme mit GUI . . . 27

6 Grundlagen 29
6.1 Der Programmkopf . . . 29
6.2 Ausgabe von Text und Zahlen . . . 30
6.3 Variablen . . . 32
6.3.1 Werte in Variablen speichern und anzeigen . . . 32
6.3.2 Inhalt einer Variable leeren . . . 33
6.3.3 Variable löschen . . . 34
6.3.4 Prüfen, ob eine Variable existiert . . . 34
6.3.5 Variabler Variablenname . . . 35
6.3.6 Variablen innerhalb eines Textes . . . 36
6.4 Fehlermeldungen verstehen . . . 38
6.5 Tastatureingabe . . . 39
6.6 Eckige Klammern . . . 40
6.7 Programm beenden . . . 40
6.8 Programm kommentieren . . . 41
6.9 Programm mit Parametern starten . . . 42

7 Zahlen, Datum und Uhrzeit 45
7.1 Zahlen . . . 45
7.1.1 Rechnen mit Zahlen . . . 45
7.1.2 Zahlen mit führender Null . . . 49

7.1.3 Zufallszahlen . . . 49
7.1.4 Eine Variable erhöhen oder reduzieren . . . 51
7.2 Datum und Uhrzeit . . . 52
7.2.1 Formatierung . . . 52
7.2.2 Rechnen mit Datum und Uhrzeit . . . 55

8 Programmablauf steuern 59
8.1 if-Bedingung . . . 59
8.2 switch-Befehl . . . 63
8.3 while-Schleife . . . 65
8.4 for-Schleife . . . 66
8.5 Schleifen vorzeitig beenden mit break und continue . . . 67

9 Prozeduren 69
9.1 Prozedur . . . 70
9.2 Prozedur auslagern . . . 77
9.3 Lokale und globale Variablen . . . 78
9.4 Upvar . . . 81
9.5 Uplevel . . . 82
9.6 Trace . . . 85

10 Programmablauf 91
10.1 Kurzform . . . 91
10.2 Details . . . 93
10.3 Ersetzungen . . . 95

11 Listen 97
11.1 Listen . . . 97
11.2 Dubletten entfernen ohne die Reihenfolge zu ändern . . . 118
11.3 Unterschied zwischen lappend und concat . . . 119
11.4 Elemente aus einer Liste extrahieren mit foreach . . . 120
11.5 Vergleich von Listen mit ::struct::set . . . 122
11.6 Alle Kombinationen aus den Elementen einer Liste (Permutation) . . . 124
11.7 Listen in Listen (eingebettete Listen) . . . 124
11.8 Listen expandieren mit {*} . . . 125

12 Text 127
12.1 Text . . . 127
12.2 Text verketten . . . 138
12.3 Text zerlegen . . . 139
12.4 Text zusammenfügen . . . 142
12.5 Text überprüfen . . . 143
12.6 Zahl als Text oder als Zahl behandeln . . . 144
12.7 Mustererkennung und Text ersetzen . . . 145
12.8 Platzhalter im Text . . . 147

13 Array 151
13.1 Array . . . 151

13.2 Array an Prozedur übergeben 156

14 Dictionary 161
14.1 Dictionary 161
14.2 Unterschied zwischen Array und Dictionary 169

15 Stack (Stapel) 171

16 Pool 177

17 Tree (Baum) 183
17.1 Tree mit Treeview kombinieren 212

18 Graph (Gerichteter Graph) 213

19 Matrix 223
19.1 Matrix kopieren 229
19.2 Matrix mittels Liste sortieren 230
19.3 csv-Datei in eine Matrix übernehmen 232
19.4 Matrix und TkTable 234

20 Dateien und Ordner 237
20.1 Dateien und Ordner 237
20.2 Ordner durchsuchen mit glob 244
20.3 Ordner durchsuchen mit fileutil::find 249
20.4 Textdatei speichern 251
20.5 Textdatei einlesen 252
20.6 Konfigurationsdatei speichern und einlesen 253
20.7 csv-Dateien öffnen bzw. erstellen 255
20.8 Binär-Dateien 258

21 Dezimale, hexadezimale, binäre und ASCII-Darstellung 263

22 Fehler während der Ausführung abfangen 265

23 Mehrsprachigkeit 267
23.1 Mehrsprachigkeit 267
23.2 Betriebssystemunabhängige Textausgabe 269

24 Programm eine Zeit lang anhalten 271

25 Andere Programme starten 273

26 Tcl mit C/C++ kombinieren 275

27 Eigene Befehle erzeugen 279

28 Formel-Eingabe durch den Anwender 281

29 Befehle aus einer Textdatei ausführen 283

30 Namensräume 285

31 Datenbank sqlite3 289

32 Objektorientierte Programmierung 297

33 Grafische Benutzerschnittstelle (GUI) 325

34 Gleichzeitige Nutzung von GUI und Konsole (nur für Windows) 327

35 Window Manager 329
35.1 Fenstertitel und Fenstergröße festlegen . . . 329
35.2 Fenstergröße abfragen . . . 331
35.3 Fenster schließen / Programm beenden . . . 331

36 Überblick über die GUI-Elemente 335

37 Geometrie-Manager 337
37.1 Pack . . . 337
37.2 Grid . . . 354
37.3 Mischen der Geometriemanager . . . 361
37.4 Place . . . 362

38 Scroll-Leisten 363
38.1 Scroll-Leisten mit pack . . . 363
38.2 Scroll-Leisten mit grid . . . 368

39 Bildschirmausgabe aktualisieren 373

40 Vorgefertigte Dialoge 375
40.1 Nachrichten-Fenster . . . 376
40.2 Dialog-Fenster . . . 377
40.3 Datei öffnen-Dialog . . . 378
40.4 Datei speichern-Dialog . . . 383
40.5 Ordner wählen-Dialog . . . 386

41 Elemente 389
41.1 Frame . . . 389
41.2 Label . . . 392
41.3 Button . . . 396
41.4 Entry . . . 400
41.5 Frame, Label, Entry und Button zusammen . . . 407
41.6 Labelframe . . . 408
41.7 Checkbutton . . . 409
41.8 Radiobutton . . . 412
41.9 Menubutton . . . 415
41.10 Spinbox . . . 416
41.11 Listbox . . . 418
41.12 Combobox . . . 427

41.13 Scale . . . 431
41.14 Progressbar . . . 433
41.15 Notebook . . . 436
41.16 Panedwindow . . . 439
41.17 Treeview . . . 444
41.17.1 Treeview mit Tree kombinieren . . . 477
41.18 Text . . . 480
41.18.1 Text-Element durchsuchen . . . 491
41.19 Calendar . . . 493

42 Weitere Elementeigenschaften 495
42.1 Einheitliche Hintergrundfarbe des Dialogs und der Elemente . . . 495
42.2 Variablen und Befehle sind global gültig . . . 496
42.3 Farben . . . 498
42.4 Schrift . . . 499
42.5 Bitmaps (nur bei klassischen Elementen) . . . 500
42.6 Bilder . . . 501
42.7 Prüfen, ob ein Element existiert . . . 503
42.8 Größe und Position eines Elements abfragen . . . 504
42.9 Element löschen . . . 505
42.10 Element deaktivieren und aktivieren . . . 506
42.11 Element ausblenden . . . 509
42.12 Element einblenden . . . 510
42.13 Element verschieben . . . 512
42.14 Fokus auf ein Element setzen . . . 513
42.15 Eigenschaften eines Elements abfragen und ändern . . . 515
42.16 Tabulatorreihenfolge der Elemente . . . 518
42.17 Automatische Größenanpassung abschalten . . . 520

43 Menü 521
43.1 Menü . . . 521
43.2 Popup-Menü . . . 528
43.3 Optionen-Menü . . . 530

44 Maus- und Tastaturereignisse (Bindings) 533
44.1 Binding für ein Element . . . 533
44.2 Binding für das gesamte Programm . . . 536
44.3 Substitution bei Bindings . . . 536
44.4 Virtuelle Ereignisse . . . 538
44.5 Alle Tastaturereignisse ignorieren . . . 540
44.6 Mausklicks eine Zeit lang ignorieren . . . 540

45 Mauszeiger 543

46 Tooltip anzeigen 547

47 Button programmseitig ausführen 549

48 Fenster oder Elemente vorübergehend sperren 551

49 Mehrere Fenster öffnen 555
49.1 Toplevel-Dialog . . . 555
49.2 Modal-Dialog . . . 557

50 TkTable 563
50.1 Maus- und Tastaturereignisse . . . 574
50.2 Selektion auswerten und Tabelle ändern . . . 576
50.3 Tabelle sortieren . . . 586

51 Canvas (Zeichenfläche) 589
51.1 Zeichenfläche und Objekte . . . 589
51.2 Objekteigenschaften ändern . . . 595
51.3 Breite und Höhe eines Objekts . . . 598
51.4 Objekt in den Vorder-/Hintergrund setzen . . . 600
51.5 Mit der Maus zeichnen . . . 602
51.6 Objekt bewegen . . . 607
51.7 Farbe eines Bildpunkts ermitteln . . . 610
51.8 Bildschirmfoto erstellen . . . 612
51.9 Zeichenfläche mit Scroll-Leisten . . . 613
51.10 Zeichenfläche, die größer als das Fenster ist . . . 614
51.11 Zeichenfläche als Hilfsmittel, viele Elemente in einem Fenster darzustellen . 617

52 Gnuplot 623

53 Animation erstellen 629

54 Layout 631
54.1 Grundlagen . . . 631
54.2 Schrift . . . 637
54.3 Farbe und Schrift der Elemente festlegen . . . 638
54.3.1 Fenster . . . 638
54.3.2 Frame . . . 639
54.3.3 Label . . . 640
54.3.4 Button . . . 641
54.3.5 Entry . . . 645
54.3.6 Labelframe . . . 649
54.3.7 Checkbutton . . . 652
54.3.8 Radiobutton . . . 655
54.3.9 Menubutton . . . 658
54.3.10 Spinbox . . . 661
54.3.11 Listbox . . . 666
54.3.12 Scrollbar . . . 668
54.3.13 Combobox . . . 672
54.3.14 Scale . . . 676
54.3.15 Progressbar . . . 678
54.3.16 Notebook . . . 680

54.3.17 Panedwindow 682
54.3.18 Treeview 683
54.3.19 Text 686

55 Mehrsprachige GUI 691

56 Client-Server-Kommunikation 695

57 Programm weitergeben 699
57.1 Weitergabe des Quellcodes 699
57.2 Weitergabe als Binärdatei 699

58 Änderungshistorie 705

1 Vorwort

Das vorliegende Buch wendet sich an Anfänger, die die Programmiersprache Tcl/Tk erlernen möchten. Es erklärt Schritt für Schritt die wichtigsten Befehle und zeigt mit über 500 Beispielen deren Verwendung. Das Buch verzichtet auf lange Texte, sondern erklärt an Hand der Beispiele, wie Sie mit Tcl/Tk programmieren. Falls Sie bereits eine ältere Version dieses Buches besitzen, finden Sie in Kapitel 58 auf Seite 705 die Änderungshistorie. Somit können Sie nachschlagen, welche Kapital ergänzt oder neu hinzugefügt wurden.

Abschließend noch ein Hinweis: Die Programm-Listings können Sie über die Internetseite `www.tcltk.info` herunterladen. Die Überschrift der Listings benennt in Klammern die Nummer der Beispiel-Datei.

2 Über Tcl/Tk

Die Sprache Tcl/Tk ist eine einfach zu erlernende, ausgereifte und stabile Programmiersprache, die es für die Betriebssysteme Windows, Mac OS und Linux gibt. Tcl/Tk-Programme sind ohne Anpassungen plattformübergreifend lauffähig. Das Sprachkonzept ist sehr einfach und schnell erlernbar, so dass man schon nach kurzer Zeit Programme mit und ohne grafischer Benutzeroberfläche schreiben kann. Die Sprache war in den 1990er Jahren weit verbreitet, erlebte aber mit dem Entstehen neuer Programmiersprachen einen Rückgang an Interesse. Dennoch ist Tcl/Tk weiterhin eine lebendige Sprache, die regelmäßig weiter entwickelt wird.

Für einen ersten Eindruck, wie kurz und verständlich Tcl/Tk-Programme sind, folgendes Beispiel: es soll ein Dialog mit grafischer Benutzeroberfläche (GUI) erscheinen, der ein Textfeld und einen OK-Button enthält. Bei einem Mausklick auf den Button wird das Programm beendet.

Listing 2.1: Ein erstes Programm (Beispiel000.tcl)

```
1 !/usr/bin/env wish
2 label .lb -text "Hallo"
3 button .bt -text "Ende" -command exit
4 pack .lb
5 pack .bt
```

2.1 Vorteile von Tcl/Tk

Tcl/Tk hat gegenüber anderen Sprachen, gerade für Anfänger, folgende Vorteile:

- es ist eine Skriptsprache mit einem einfachen Sprachkonzept
- man kann sowohl prozedural als auch objektorientiert programmieren
- der Programmieranfänger braucht sich nicht in eine Integrierte Entwicklungsumgebung (IDE) einarbeiten
- Programme mit grafischer Benutzerschnittstelle (GUI) können direkt in Tcl/Tk erstellt werden, ohne dass die GUI zusätzlich installiert und eingebunden werden muss
- einfacher Zugriff auf sqlite, eine SQL-Datenbank
- deutsche Umlaute und Sonderzeichen sind direkt verfügbar

- die Programme sind sehr leicht mehrsprachig zu gestalten
- Tcl/Tk gibt es für alle gängigen Betriebssysteme (Windows, Mac OS, Linux)

2.2 Hilfreiche Internetseiten zu Tcl/Tk

Im folgenden sind einige hilfreiche Internetseiten aufgelistet.

Tabelle 2.1: Internetseiten

Internetseite	Beschreibung
`www.tcl.tk/man`	Befehlsreferenz für Tcl/Tk
`www.tkdocs.com`	Tk-Befehle mit Beispielen
`https://core.tcl-lang.org/tcllib/doc/trunk/embedded/md/toc.md`	Dokumentation weiterer Tcl-Befehle
`https://core.tcl-lang.org/tklib/doc/trunk/embedded/www/toc.html`	Dokumentation weiterer TK-Befehle
`www.stackoverflow.com/questions/tagged/tcl`	Englischsprachiges Tcl/Tk-Forum
`http://wiki.tcl.tk`	Tcl/Tk-Wiki
`http://wiki.tcl-lang.org/405`	Beschreibung, welche Änderungen in jeder Version von Tcl/Tk gemacht wurden

2.3 Bücher zu Tcl/Tk

Das sehr lehrreiche und uneingeschränkt empfehlenswerte Standardwerk stammt vom Erfinder der Tcl/Tk-Sprache John Ousterhout:

- John K. Ousterhout, Ken Jones und Eric Foster-Johnson: Tcl and the Tk Toolkit, Addison-Wesley Longman (4. September 2009)

Weitere Bücher sind:

- Ashok P. Nadkarni: The Tcl Programming Language: A Comprehensive Guide (18. Juli 2017)
- Clif Flynt: Tcl/Tk: A Developer's Guide, Morgan Kaufmann (15. März 2012)

Eine sehr gute Darstellung der objektorientierten Programmierung mit Tcl/Tk findet man online bei

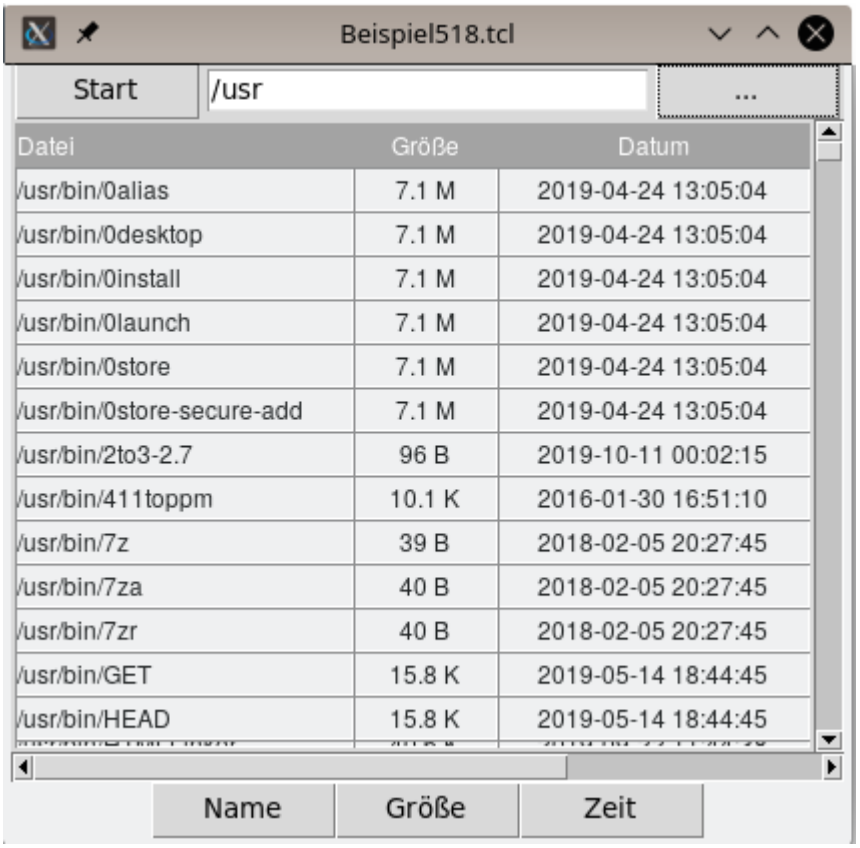
- Ashok P. Nadkarni: Object Oriented Programming in Tcl

Und noch zwei ältere Bücher:

- Eric Foster-Johnson: Graphical Applications with Tcl and Tk, Second Edition, M and T Books (1997)
- Mark Harrison, Michael McLennan: Effective Tcl/Tk Programming, Addison-Wesley Longman (1998)

3 Beispiel-Programm

Bevor man sich näher mit einer Programmiersprache befasst, ist es eventuell interessant, sich einmal den Programmcode anhand eines praktischen Beispiels näher anzuschauen. Es kommt dabei nicht darauf an, jedes Details zu verstehen. Vielmehr bekommt man einen Eindruck, ob einem die Syntax liegt und ob der Code einem weitgehend verständlich erscheint. Deshalb möchte ich Ihnen das nachstehende Programm zeigen.



Wenn der Anwender das Programm startet, werden alle Dateien im Startordner sowie in allen Unterordnern eingelesen. Dabei werden auch die Dateieigenschaften `Größe` und `Datum/Uhrzeit` ermittelt. Die Dateien werden mit Ihren Eigenschaften tabellarisch dargestellt. Über drei Buttons kann der Anwender die Dateiliste nach Name, Größe und Datum/Uhrzeit sortieren, jeweils absteigend und aufsteigend im Wechsel. Über einen Button kann er einen Dialog öffen, um den Startordner zu wechseln. Wenn der Anwender den Startordner manuell im Eingabefeld eingibt, wird der Suchvorgang durch die Return-Taste oder einen Mausklick auf den `Start`-Button ausgelöst.

Der Programmcode ist kommentiert. Die Kommentarzeilen fangen mit einem #-Zeichen an. Aus drucktechnischen Gründen werden in den Programmcodes dieses Buchs die deutschen Umlaute (z. B. ä) und Sonderzeichen (z. B. ß) ersetzt (z. B. durch `ae` oder `ss`).

Listing 3.1: Beispiel-Programm (Beispiel518.tcl)

```
#!/usr/bin/env tclsh

package require fileutil
package require Tktable

proc Sortieren {tmpVar Zeilen Spalten Sortierspalte Sortierung Art} {
  # Sortiert die Tabelle
  # Da die Tabellenvariable ein Array ist und
  # man ein Array nicht sortieren kann, wird
  # das Array in eine Liste umgewandelt,
  # dann die Liste sortiert und schliesslich
  # die Liste in ein Array zurueck umgewandelt.
  #
  # tmpVar = Name der Tabellen-Variable
  # Zeilen = Anzahl der Zeilen in der Tabelle
  # Spalten = Anzahl der Spalten in der Tabelle
  # Sortierspalte = Index der Spalte, nach der sortiert wird
  # Sortierung = auf- oder absteigend sortieren
  # Art = numerisch oder alphanumerisch sortieren

  # Verweis auf die Tabellen-Variable
  upvar $tmpVar Tabelle

  # Programm fuer weitere Aktionen sperren
  tk busy hold .
  tk busy configure . -cursor watch
  update

  # Tabellen-Variable (Array) in Liste umwandeln (ohne Titelzeile)
  set Liste {}
  for {set i 1} {$i < $Zeilen} {incr i} {
    # Alle Werte in der Zeile zu einer Liste zusammenfuegen
    set ListeZeilenwerte {}
    for {set j 0} {$j < $Spalten} {incr j} {
      lappend ListeZeilenwerte $Tabelle($i,$j)
    }
    # Die Liste mit den Zeilenwerten der (Gesamt-)Liste hinzufuegen.
    # Es handelt sich also um Listen innerhalb einer Liste.
    lappend Liste $ListeZeilenwerte
  }

  # Liste sortieren
  set Liste [lsort $Art $Sortierung -index $Sortierspalte $Liste]

```

```
  # Liste in Tabellen-Variable umwandeln
  for {set i 1} {$i < $Zeilen} {incr i} {
    # weil die Liste mit Index=0 startet, die Werte aber in Zeile 1 der
    # Tabelle angezeigt werden sollen, muss der Index der Liste
    # gegenueber der Tabelle um 1 reduziert werden.
    set k [expr $i - 1]
    for {set j 0} {$j < $Spalten} {incr j} {
      # Die innerste Klammer holt die gesamte Zeile mit den Werten,
      # die aeussere Klammer holt aus dieser Zeile die Werte zu
      # jeder Spalte.
      set Tabelle($i,$j) [lindex [lindex $Liste $k] $j]
    }
  }

  # Sortierung wechseln
  if {$Sortierung eq "-increasing"} {
    set Sortierung "-decreasing"
  } else {
    set Sortierung "-increasing"
  }

  # Sperrung des Programms aufheben
  tk busy forget .
  update

  return $Sortierung
}

proc DateienEinlesenVergleich {Datei} {
  # Definiert, welche Dateien eingelesen werden sollen.
  # In diesem Beispiel sollen alle Dateien eingelesen werden.
  return [string match * $Datei]
}

proc DateienEinlesen {tmpVar Ordner} {
  # Liest die Dateien ein

  # Verweis auf die Tabellen-Variable
  upvar $tmpVar Tabelle

  # Prueft, ob der Ordner existiert
  if {![file exists $Ordner]} {return}

  # Programm fuer weitere Aktionen sperren
  tk busy hold .
  tk busy configure . -cursor watch
  update

```

```
93    # Tabelleninhalt loeschen
94    array unset Tabelle
95
96    # Tabellenueberschrift
97    set Tabelle(0,0) "Datei"
98    set Tabelle(0,1) "Groesse"
99    set Tabelle(0,2) "Datum"
100
101   # Dateien einlesen
102   set Dateien [fileutil::find $Ordner ↩
         DateienEinlesenVergleich]
103
104   # Dateien mit den Dateieigenschaften in die Tabelle ↩
         uebernehmen
105   set Zeile 1
106   foreach Datei $Dateien {
107     if {[file isfile $Datei]} {
108       set Groesse [file size $Datei]
109       if {$Groesse >= 1048576} {
110         set FormatierteGroesse "[expr round(double(↩
               $Groesse)/1048576.0*10.0)/10.0] M"
111       } elseif {$Groesse >= 1024} {
112         set FormatierteGroesse "[expr round(double(↩
               $Groesse)/1024.0*10.0)/10.0] K"
113       } else {
114         set FormatierteGroesse "$Groesse B"
115       }
116
117       set Zeit [file mtime $Datei]
118       set FormatierteZeit [clock format $Zeit -format {%Y-↩
             %m-%d %H:%M:%S}]
119
120       set Tabelle($Zeile,0) $Datei
121       set Tabelle($Zeile,1) $FormatierteGroesse
122       set Tabelle($Zeile,2) $FormatierteZeit
123       set Tabelle($Zeile,3) $Groesse
124       set Tabelle($Zeile,4) $Zeit
125       incr Zeile
126     }
127   }
128
129   # Zeilenanzahl der Tabelle anpassen
130   .frTabelle.tbTabelle configure -rows $Zeile
131
132   # Sperrung des Programms aufheben
133   tk busy forget .
134   update
135 }
136
137 proc OrdnerWaehlen {Ordner} {
138   # Auswahl des Startordners
139   set tmp [tk_chooseDirectory]
140   if {$tmp ne ""} {
```

```
    set Ordner $tmp
  }
  return $Ordner
}

# Ordnerauswahl erzeugen und anzeigen
set Startordner $env(HOME)
ttk::frame .frOrdner
ttk::entry .frOrdner.enOrdner -width 10 -textvariable Startordner
ttk::button .frOrdner.btOrdner -text "..." -command {
  set Startordner [OrdnerWaehlen $Startordner]
  .frOrdner.btStart invoke
}
ttk::button .frOrdner.btStart -text "Start" -command {DateienEinlesen Tabelle $Startordner}
pack .frOrdner.btStart -side left -padx 2
pack .frOrdner.enOrdner -side left -padx 2 -fill x -expand yes
pack .frOrdner.btOrdner -side left -padx 2
pack .frOrdner -side top -fill x

# Return-Taste mit Start-Button verbinden.
bind .frOrdner.enOrdner <Return> {.frOrdner.btStart invoke}

# Tabelle definieren
ttk::frame .frTabelle
table .frTabelle.tbTabelle \
  -titlerows 1 \
  -titlecols 0 \
  -cols 3 \
  -rows 1 \
  -height 20 \
  -width 3 \
  -rowheight 1 \
  -colwidth 15 \
  -maxheight 300 \
  -maxwidth 600 \
  -xscrollcommand {.frTabelle.xscroll set} \
  -yscrollcommand {.frTabelle.yscroll set} \
  -variable Tabelle

# Interne Spaltenbezeichnung definieren
.frTabelle.tbTabelle tag col SpalteName 0
.frTabelle.tbTabelle tag col SpalteGroesse 1
.frTabelle.tbTabelle tag col SpalteZeit 2

# Spaltenbreite definieren
.frTabelle.tbTabelle width 0 50
.frTabelle.tbTabelle width 1 10
.frTabelle.tbTabelle width 2 50

```

```
190 # Spaltenausrichtung definieren
191 .frTabelle.tbTabelle tag configure SpalteName -anchor w
192 .frTabelle.tbTabelle tag configure SpalteGroesse -anchor ↩
    center
193 .frTabelle.tbTabelle tag configure SpalteZeit -anchor ↩
    center
194
195 # Scroll-Leisten
196 ttk::scrollbar .frTabelle.xscroll -command {↩
    .frTabelle.tbTabelle xview} -orient horizontal
197 ttk::scrollbar .frTabelle.yscroll -command {↩
    .frTabelle.tbTabelle yview} -orient vertical
198
199 # Tabelle anzeigen
200 pack .frTabelle -side top -fill both -expand yes
201 pack .frTabelle.xscroll -side bottom -fill x
202 pack .frTabelle.yscroll -side right -fill y
203 pack .frTabelle.tbTabelle -side top -fill both -expand yes
204
205 # Buttons erzeugen und anzeigen
206 ttk::frame .frButton
207 ttk::button .frButton.btName -text "Name" -command {set ↩
    NameSortierung [Sortieren Tabelle [.frTabelle.tbTabelle↩
     cget -rows] 5 0 $NameSortierung "-dictionary"]}
208 ttk::button .frButton.btGroesse -text "Groesse" -command {↩
    set GroesseSortierung [Sortieren Tabelle [↩
    .frTabelle.tbTabelle cget -rows] 5 3 $GroesseSortierung↩
     "-integer"]}
209 ttk::button .frButton.btZeit -text "Zeit" -command {set ↩
    ZeitSortierung [Sortieren Tabelle [.frTabelle.tbTabelle↩
     cget -rows] 5 4 $ZeitSortierung "-integer"]}
210 pack .frButton.btName -side left
211 pack .frButton.btGroesse -side left
212 pack .frButton.btZeit -side left
213 pack .frButton -side bottom
214
215 # Sortierung (auf-/absteigend) initialisieren
216 set NameSortierung "-increasing"
217 set GroesseSortierung "-decreasing"
218 set ZeitSortierung "-decreasing"
219
220 # Start-Button aufrufen
221 .frOrdner.btStart invoke
```

4 Installation

Im Folgenden wird die Installation von Tcl/Tk unter Windows und Linux beschrieben. Tcl/Tk ist außerdem auch für OS X erhältlich.

4.1 Installation unter Windows

Installieren Sie Tcl/Tk von der Seite http://www.magicsplat.com/tcl-installer/index.html. Installieren Sie außerdem einen Editor wie z. B. Geany (http:///wwww.geany.org). Da Geany zusätzlich GTK+ benötigt, laden Sie das Programm am besten von folgender Seite herunter: http://www.heise.de/download/geany-1130597.html

4.2 Installation unter Linux

Installieren Sie aus dem Repository ihrer Distribution tcl8.5 und tk8.5 (oder neuer). Außerdem sollten Sie

- tcllib
- tklib
- libtk-img
- tk-table
- sqlite3
- libsqlite3-tcl

installieren. Installieren Sie zusätzlich den Editor geany.

5 Tcl/Tk-Programme starten

5.1 Programme unter Windows starten

Beim Ausführen eines Tcl-Programms unter Windows muss man unterscheiden, ob das Programm mit oder ohne grafische Benutzeroberfläche (GUI) arbeitet. Dies liegt daran, dass Windows nicht automatisch erkennt, ob das Tcl-Programm in einer Konsole läuft oder eine GUI benutzt, sondern jedes Tcl-Programm standardmäßig als GUI-Programm startet.

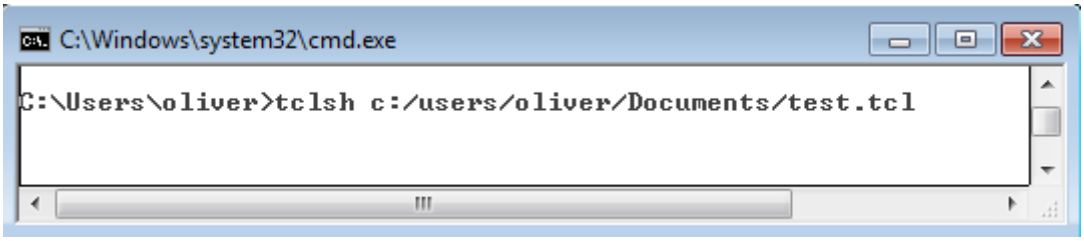
5.1.1 Programme ohne GUI

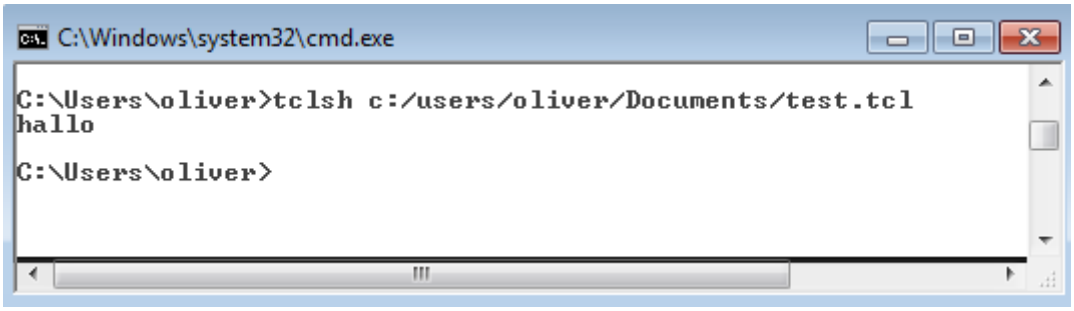
Starten Sie einen Texteditor (z.B. Geany). Geben Sie nun folgende Zeilen ein (Listing 5.1):

Listing 5.1: Test ohne GUI (Beispiel000b.tcl)

```
1 #!/usr/bin/env tclsh
2 puts hallo
```

Speichern Sie das Programm unter `test.tcl`. Starten Sie über das Startmenü von Windows eine Command-Shell (Befehl cmd), und geben Sie dann den Befehl `tclsh` gefolgt vom Namen des Tcl-Programms (inklusive des gesamten Pfads) ein. Jetzt startet das Tcl-Programm in der Konsole.





Wenn Sie im Umgang mit der Konsole nicht vertraut sind, sollten Sie im Internet nach einer Anleitung für `cmd.exe` suchen. Die Tabelle 5.1 zeigt die wichtigsten Befehle in der Konsole.

Tabelle 5.1: Befehle in der Konsole

Befehl	Beschreibung
c:	wechselt auf die Festplatte C
md Verzeichnis	erstellt ein Verzeichnis
cd Verzeichnis	wechselt in das Verzeichnis
cd ..	wechselt in das höhere Verzeichnis
cls	löscht den Bildschirm
dir *.*	zeigt alle Dateien im Verzeichnis
del Datei	löscht eine Datei

5.1.2 Programme mit GUI

Wenn das Tcl-Programm eine GUI hat, brauchen Sie nur einen Doppelklick auf die Datei zu machen. Geben Sie im Texteditor (z. B. Geany) folgendes Programm ein (Listing 5.2):

Listing 5.2: Test mit GUI (Beispiel000c.tcl)

```
1 #!/usr/bin/env wish
2 label .lb -text Hallo
3 button .bt -text OK -command exit
4 pack .lb
5 pack .bt
```

Speichern Sie das Programm unter `test.tcl` und starten Sie es mit einem Doppelklick.

5.2 Programme unter Linux starten

5.2.1 Programme ohne GUI

Starten Sie einen Texteditor (z.B. Geany)und geben Sie folgende Zeilen ein (Listing 5.3):

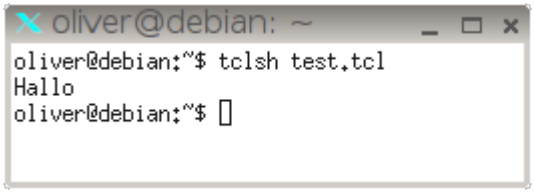
Listing 5.3: Test ohne GUI (Beispiel000b.tcl)

```
1 #!/usr/bin/env tclsh
2 puts hallo
```

Speichern Sie das Programm unter `test.tcl`. Sie können das Tcl-Programm auf zwei Arten starten.

Programm mit tclsh starten

Öffnen Sie ein Terminal-/Konsolenfenster und geben Sie den Befehl `tclsh test.tcl` ein.



Programm mit dem Programmnamen starten

Öffnen Sie ein Terminal-/Konsolenfenster und geben Sie den Befehl `chmod u+x test.tcl` ein. Damit machen Sie das Programm direkt ausführbar. Starten Sie das Programm aus dem Terminal-/Konsolenfenster heraus mit dem Befehl `./test.tcl`.

5.2.2 Programme mit GUI

Geben Sie im Texteditor (z. B. Geany) folgendes Programm ein (Listing 5.4):

Listing 5.4: Test mit GUI (Beispiel000c.tcl)

```
#!/usr/bin/env wish
label .lb -text Hallo
button .bt -text OK -command exit
pack .lb
pack .bt
```

Speichern Sie das Programm unter `test.tcl`. Auch jetzt gibt es wieder zwei Varianten, das Programm zu starten.

Programm mit wish starten

Öffnen Sie ein Terminal-/Konsolenfenster und geben Sie den Befehl `wish test.tcl` ein.

Programm mit dem Programmnamen starten

Öffnen Sie ein Terminal-/Konsolenfenster und geben Sie den Befehl `chmod u+x test.tcl` ein. Damit machen Sie das Programm direkt ausführbar. Starten Sie das Programm aus dem Terminal-/Konsolenfenster heraus mit dem Befehl `./test.tcl`.

6 Grundlagen

6.1 Der Programmkopf

Jedes Programm ohne grafische Benutzerschnittstelle beginnt mit

```
#!/usr/bin/env tclsh
```

und jedes Programm mit grafischer Benutzerschnittstelle (GUI) mit

```
#!/usr/bin/env wish
```

Ganz wichtig: es darf kein Leerzeichen am Anfang dieser Zeile stehen! Wenn man eine bestimmte Tcl/Tk-Version verwenden möchte, kann man auch die Versionsnummer angeben. Die erste Zeile sieht dann wie folgt aus:

```
#!/usr/bin/env tclsh8.5
```

bzw.

```
#!/usr/bin/env wish8.5
```

Generell gilt für Tcl/Tk:

- Jeder Befehl steht in einer neuen Zeile oder ist mit einem Semikolon abgeteilt.
- Die einzelnen Teile eines Befehls werden durch Leerzeichen getrennt.
- Die Befehle werden von oben nach unten abgearbeitet.
- Tcl/Tk unterscheidet zwischen Groß- und Kleinbuchstaben.
- Das Programm darf Leerzeilen enthalten.
- Wenn ein Befehl über mehrere Zeilen geht, muss man am Zeilenende ein \ Zeichen setzen.
- Kommentare werden mit einem # vom Programmcode getrennt.

Listing 6.1: Grundlagen (Beispiel000d.tcl)

```
#!/usr/bin/env tclsh

# Das ist eine Kommentarzeile.
# Sie beginnt mit einem # Zeichen.

# Ein Befehl

puts Hallo

```

```
10 # Zwei Befehle in einer Zeile werden mit einem
11 # Semikolon getrennt
12
13 puts Hallo; puts Hallo
14
15 # Wenn ein Kommentar in derselben Zeile steht
16 # wie der Befehl, wird der Kommentar mit
17 # einem Semikolon und einem # Zeichen abgegrenzt
18
19 puts Hallo; # Kommentar
20
21 # Wenn ein Befehl ueber mehrere Zeilen geht,
22 # muss ein \ am Zeilenende stehen
23
24 puts "Hallo, wie \
25 geht es Ihnen?"
```

6.2 Ausgabe von Text und Zahlen

Befehle:

- `puts Text`
- `puts "Text mit Leerzeichen"`
- `puts -nonewline "Text"`
- `flush stdout`

Der Befehl `puts` gibt Text oder eine Zahl aus. Wenn der Text Leerzeichen enthält, muss der Text in doppelte Anführungszeichen gesetzt werden. Man sollte sich angewöhnen, Texte generell in Anführungszeichen zu setzen. Wenn man innerhalb eines Textes Anführungszeichen verwenden möchte, muss man sie mit einem vorangestellten Schrägstrich \ maskieren. Normalerweise sendet der `puts`-Befehl automatisch ein Zeilenende-Signal, so dass die nächste Ausgabe in einer neuen Zeile erfolgt. Wenn man aber in derselben Zeile weiteren Text ausgeben möchte, fügt man die Option `-nonewline` hinzu. Der Text wird solange in derselben Zeile fortgeführt, bis man mit dem Befehl `flush stdout` die Ausgabe beendet.

Listing 6.2: Ausgabe von Text und Zahlen (Beispiel001.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts Hallo
4 puts "Wie geht es dir?"
5 puts 123
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_001.tcl
Hallo
Wie geht es dir?
123
oliver@debian:~$
```

In Zeile 3 wird ein einzelnes Wort ausgegeben. Da es kein Leerzeichen enthält, kann man auf Anführungszeichen verzichten. In Zeile 4 wird ein Text mit Leerzeichen ausgegeben. Er muss in Anführungszeichen gesetzt werden. In Zeile 5 wird eine Zahl ausgegeben.
Wenn im Text Anführungszeichen enthalten sind, müssen sie mit \ maskiert werden, damit sie nicht als beendende Anführungszeichen des `puts`-Befehls interpretiert werden (Listing 6.3).

Listing 6.3: Anführungszeichen im Text (Beispiel002.tcl)

```
1  #!/usr/bin/env tclsh
2
3  # Wenn im Text ein Anfuehrungszeichen angezeigt
4  # werden soll, muss man vor das Anfuerungszeichen
5  # ein \ schreiben
6
7  puts "Das Buch \"Tcl fuer Anfaenger\" hat 500 Seiten."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_002.tcl
Das Buch "Tcl für Anfänger" hat 500 Seiten.
oliver@debian:~$
```

Will man mehrere Textausgaben in derselben Zeile machen, muss man verhindern, dass automatisch ein Zeilenende ausgegeben wird. Dazu verwendet man die Option `-nonewline`. Wenn schließlich die Zeile fertig zusammengesetzt ist, gibt man sie mit dem Befehl `flush stdout` auf den Bildschirm aus (Listing 6.4).

Listing 6.4: Textzeile zusammensetzen (Beispiel368.tcl)

```
1  #!/usr/bin/env tclsh
2
3  puts "a"
4  puts "b"
5  puts "c"
6  puts "d"
7
8  puts -nonewline "a"
9  puts -nonewline "b"
10 puts -nonewline "c"
11 puts -nonewline "d"
12 flush stdout
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_368.tcl
a
b
c
d
abcd
oliver@debian:~$
```

In den Zeilen 3 bis 6 werden die Buchstaben einzelnen Zeilen ausgegeben. In den Zeilen 8 bis 12 werden ebenfalls die vier Buchstaben ausgegeben, allerdings in derselben Zeile. Durch die Option `-nonewline` und den abschliessenden Befehl `flush stdout` werden alle Buchstaben in derselben Zeile angezeigt.

6.3 Variablen

6.3.1 Werte in Variablen speichern und anzeigen

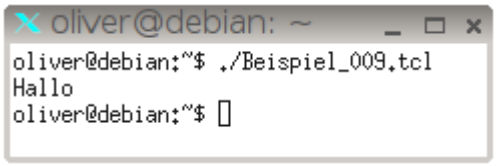
Befehle:

- `set Variable Wert`
- `puts $Variable`

Eine Variable ist wie ein Kästchen, in das man etwas hineinlegt (meistens Buchstaben, Texte oder Zahlen) und wieder herausholen kann. Der Name einer Variablen darf nur aus Buchstaben, Ziffern und dem Unterstrich bestehen. Er muss mit einem Buchstaben beginnen. Es dürfen keine Sonderzeichen wie z. B. `§%äöüß` im Namen verwendet werden. Mit dem Befehl `set` speichert man einen Wert in einer Variablen. Mit dem Dollarzeichen `$` vor der Variablen gibt man den Wert der Variablen wieder aus. Verwenden Sie sprechende Variablennamen. Es ist besser die Eingabe des Anwenders in einer Variablen namens Eingabe oder Antwort zu speichern als in einer Variablen namens A12Z. Die Variablen werden bei Tcl/Tk im Unterschied zu vielen anderen Programmiersprachen nicht typisiert, d. h. sie werden nicht zuerst als String, Integer, Float usw. definiert. Tcl/Tk speichert intern alles als Text ab und typisiert die Variable automatisch je nach Verwendung.
Das Listing 6.5 zeigt, wie man Text in einer Variablen speichert und den Inhalt der Variable wieder ausgibt.

Listing 6.5: Text in einer Variablen speichern und anzeigen (Beispiel009.tcl)

```
1 #!/usr/bin/env tclsh
2 set MeineVariable Hallo
3 puts $MeineVariable
```



Auf die gleiche Weise kann auch eine Zahlen in einer Variablen gespeichert werden (Listing 6.6).

Listing 6.6: Zahl in einer Variablen speichern und anzeigen (Beispiel010.tcl)

```
1 #!/usr/bin/env tclsh
2 set MeineVariable 123
3 puts $MeineVariable
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_010.tcl
123
oliver@debian:~$
```

Das Listing 6.7 zeigt, dass man in dieselbe Variable zuerst Text und danach eine Zahl speichern kann, ohne dass es (wie teilweise bei anderen Programmiersprachen) zu einem Fehler kommt. Tcl/Tk typisiert die Variable automatisch je nach Inhalt.

Listing 6.7: Automatische Typisierung (Beispiel011.tcl)

```
#!/usr/bin/env tclsh
set MeineVariable Hallo
puts $MeineVariable
set MeineVariable 123
puts $MeineVariable
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_011.tcl
Hallo
123
oliver@debian:~$
```

6.3.2 Inhalt einer Variable leeren

Befehle:

- `set Variable ""`
- `set Variable {}`

Um den Inhalt einer Variablen zu leeren, weist man der Variablen eine leere Menge zu. Bei normalen Variable verwendet man den Befehl `set Variable ""`, bei Listen den Befehl `set Variable {}`.

Listing 6.8: Eine Variable leeren (Beispiel012.tcl)

```
#!/usr/bin/env tclsh

set MeineVariable Hallo
puts "Inhalt 1: $MeineVariable"

set MeineVariable ""
puts "Inhalt 2: $MeineVariable"

set MeineListe {Montag Dienstag Mittwoch}
puts "Inhalt 3: $MeineListe"

set MeineListe {}
puts "Inhalt 4: $MeineListe"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_012.tcl
Inhalt 1: Hallo
Inhalt 2:
Inhalt 3: Montag Dienstag Mittwoch
Inhalt 4:
oliver@debian:~$
```

In Zeile 6 wird der Inhalt der Variable `MeineVariable` geleert. Die Variable existiert aber weiterhin, so dass der (leere) Inhalt ausgegeben werden kann (Zeile 7). In Zeile 12 wird die Listen-Variable `MeineListe` geleert und in Zeile 13 ausgegeben.

6.3.3 Variable löschen

Befehl:

- `unset Variable`

Der Befehl `unset` löscht eine Variable. Die Variable existiert danach nicht mehr.

Listing 6.9: Eine Variable löschen (Beispiel013.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set MeineVariable Hallo
4 puts $MeineVariable
5
6 unset MeineVariable
7 puts $MeineVariable
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_013.tcl
Hallo
can't read "MeineVariable": no such variable
    while executing
"puts $MeineVariable"
    (file "./Beispiel_013.tcl" line 6)
oliver@debian:~$
```

In Zeile 6 wird die Variable `MeineVariable` wird gelöscht, so dass sie nicht mehr existiert. Wenn man die Variable danach ausgibt (Zeile 7), kommt es zu einer Fehlermeldung.

6.3.4 Prüfen, ob eine Variable existiert

Befehl:

- `[info exists Variable]`

Der Befehl `info exists` prüft, ob eine Variable existiert.

Listing 6.10: Prüfen, ob eine Variable existiert (Beispiel510.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set a 1
4
5  if {[info exists a]} {
6          puts "Die Variable a existiert"
7  } else {
8          puts "Die Variable a existiert nicht"
9  }
10
11 if {[info exists b]} {
12         puts "Die Variable b existiert"
13 } else {
14         puts "Die Variable b existiert nicht"
15 }
```

```
oli@debian: ~
oli@debian:~$ ./Beispiel510.tcl
Die Variable a existiert
Die Variable b existiert nicht
oli@debian:~$
```

In Zeile 3 wird die Variable `a` mit dem Wert 1 definiert. Eine zweite Variable `b` wird nicht erzeugt. In der Zeile 5 wird geprüft, ob die Variable `a` existiert (der `if`-Befehl wird in Kapitel 8.1 vorgestellt). Ebenso wird in Zeile 11 geprüft, ob eine Variable `b` existiert.

6.3.5 Variabler Variablenname

Man kann einen Variablennamen bilden, indem man den Inhalt einer anderen Variablen als Teil des Namens oder sogar als ganzen Namen einsetzt.

Listing 6.11: Variabler Variablenname (Beispiel381.tcl)

```
#!/usr/bin/env tclsh

set MeineVariable Spieler1
set $MeineVariable 100

# Der Variablen MeineVariable wird die Zahl 2 zugewiesen.

set MeineVariable 2
set Spieler$MeineVariable 200

puts "Spieler 1 hat $Spieler1 Punkte."
puts "Spieler 2 hat $Spieler2 Punkte."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_381.tcl
Spieler 1 hat 100 Punkte.
Spieler 2 hat 200 Punkte.
oliver@debian:~$
```

In Zeile 3 wird der Variablen `MeineVariable` der Text `Spieler1` zugewiesen. In Zeile 4 wird beim Ausführen des Programms zuerst die Variable `MeineVariable` durch deren Inhalt `Spieler1` ersetzt. Danach wird der Wert `100` zugewiesen. Der Befehl sieht letztlich wie folgt aus: `set Spieler1 100`. In Zeile 8 wird der Variablen `MeineVariable` die Zahl `2` zugewiesen. In Zeile 9 wird die Variable `MeineVariable` durch deren Inhalt `2` ersetzt. Danach wird der Wert `200` zugewiesen. Der Befehl sieht somit wie folgt aus: `set Spieler2 200`. In den Zeilen 11 und 12 werden die Inhalte der beiden Variablen `Spieler1` und `Spieler2` ausgegeben.

6.3.6 Variablen innerhalb eines Textes

Wenn eine Variable innerhalb eines Textes ausgegeben werden soll, muss man darauf achten, dass die Variable vom restlichen Text abgegrenzt ist. Die Abgrenzung erfolgt entweder durch ein Leerzeichen, Komma, Punkt oder `$`-Zeichen nach der Variablen oder durch geschweifte Klammern `{}` um die Variable herum.

Listing 6.12: Nicht abgegrenzte Variable (Beispiel016.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Tag "Mon"
4 puts "Heute ist $Tagtag."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_016.tcl
can't read "Tagtag": no such variable
    while executing
"puts "Heute ist $Tagtag.""
    (file "./Beispiel_016.tcl" line 4)
oliver@debian:~$
```

Eine vom restlichen Text nicht abgegrenzte Variable führt zu einer Fehlermeldung. Die Variable `Tag` ist nicht vom restlichen Textteil `tag` abgegrenzt, so dass der Interpreter nach der Variablen `Tagtag` sucht. Da es diese Variable nicht gibt, kommt es zu einer Fehlermeldung.

Listing 6.13: Abgegrenzte Variable (Beispiel017.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Tag "Mon"
4 puts "Heute ist ${Tag}tag."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_017.tcl
Heute ist Montag.
oliver@debian:~$
```

Die Variable `Tag` wurde in geschweifte Klammern `{}` gesetzt und ist dadurch vom restlichen Text abgetrennt. Somit kommt es zu keiner Fehlermeldung.

Listing 6.14: Abgrenzung mit Komma, Punkt, Semikolon (Beispiel018.tcl)

```
#!/usr/bin/env tclsh

set Tag "Dienstag"
puts "Montag,$Tag,Mittwoch."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_018.tcl
Montag,Dienstag,Mittwoch.
oliver@debian:~$
```

Das Komma direkt hinter der Variablen `$Tag` grenzt den Variablennamen ab, so dass es zu keiner Fehlermeldung kommt. Ebenso grenzen Punkt und Semikolon die Variable vom Text ab.

Listing 6.15: Mehrere Variablen direkt hinter einander (Beispiel019.tcl)

```
#!/usr/bin/env tclsh

set Teil1 "Computer"
set Teil2 "kurs"
puts $Teil1$Teil2
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_019.tcl
Computerkurs
oliver@debian:~$
```

Auch im Listing 6.15 kommt es zu keinem Fehler, weil das Dollarzeichen `$` die zwei Variablen trennt.

Listing 6.16: Variablen-Namen im Text ausgeben (Beispiel020.tcl)

```
#!/usr/bin/env tclsh

set Zahl 3
puts "Der Befehl \"puts \$Zahl\" hat als Ergebnis die Zahl
    $Zahl."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_020.tcl
Der Befehl "puts $Zahl" hat als Ergebnis die Zahl 3.
oliver@debian:~$
```

Mit einem Schrägstrich \ maskiert man das nächste Zeichen, so dass es vom Tcl-Interpreter nicht interpretiert wird. Das ist in Zeile 4 dreimal der Fall: Zuerst wird das Anführungszeichen maskiert, damit es nicht als Ende des Befehls `puts` interpretiert wird. Danach wird die Variable `$Zahl` maskiert, damit nicht der Inhalt der Variablen angezeigt wird, sondern der Name der Variablen. Und schließlich wird erneut das Anführungszeichen maskiert.

6.4 Fehlermeldungen verstehen

Wenn man ein Programm schreibt, macht man fast immer auch den einen oder anderen Fehler. Der Tcl-Interpreter erkennt beim Ausführen die syntaktischen Fehler und zeigt sie an. Die Fehlermeldung gibt meistens einen genauen Hinweis auf die Art des Fehlers und die Stelle im Programm.

Listing 6.17: Fehlermeldung (Beispiel439.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts $Zahl
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_439.tcl
can't read "Zahl": no such variable
    while executing
"puts $Zahl"
    (file "./Beispiel_439.tcl" line 3)
oliver@debian:~$
```

Die Fehlermeldung besagt, dass es die Variable `Zahl` nicht gibt. Der Fehler ist aufgetaucht, als der Befehl `puts $Zahl` ausgeführt wurde. Der Fehler befindet sich in Zeile 3.

Listing 6.18: Fehlermeldung in einer Prozedur (Beispiel440.tcl)

```
1 #!/usr/bin/env tclsh
2
3 proc Ausgabe {} {
4         puts $Zahl
5 }
6
7 Ausgabe
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_440.tcl
can't read "Zahl": no such variable
    while executing
"puts $Zahl"
    (procedure "Ausgabe" line 2)
    invoked from within
"Ausgabe"
    (file "./Beispiel_440.tcl" line 7)
oliver@debian:~$
```

Auch wenn Prozeduren erst in Kapitel 9 auf Seite 69 behandelt werden, soll an dieser Stelle die Fehlermeldung analysiert werden. Die Fehlermeldung zeigt, dass die Variable `Zahl` unbekannt ist. Der Fehler ist aufgetaucht, als der Befehl `puts $Zahl` ausgeführt wurde. Der Fehler ist in Zeile 2 der Prozedur Ausgabe entstanden. Man beachte, dass die Zeilennummer sich nicht auf das gesamte Programm bezieht, sondern auf die Prozedur. Die Prozedur wurde durch den Befehl `Ausgabe` in Zeile 7 aufgerufen.

6.5 Tastatureingabe

Befehle:

- `gets stdin`
- `set MeineVariable [gets stdin]`
- `puts -nonewline "Text"`
- `flush stdout`

Der Befehl `gets stdin` übernimmt die Tastatureingaben. In Kombination mit `set Variable` kann man die Tastatureingabe speichern.

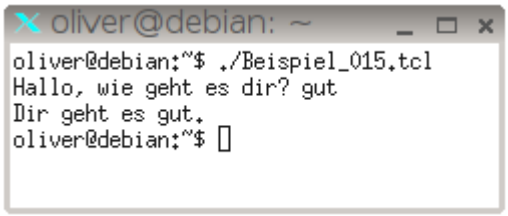
Listing 6.19: Tastatureingabe (Beispiel014.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts "Hallo, wie geht es dir?"
4 set Antwort [gets stdin]
5 puts "Dir geht es $Antwort."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_014.tcl
Hallo, wie geht es dir?
gut
Dir geht es gut.
oliver@debian:~$
```

Listing 6.20: Textausgabe und Tastatureingabe in einer Zeile (Beispiel015.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts -nonewline "Hallo, wie geht es dir? "
4 flush stdout
5 set Antwort [gets stdin]
6 puts "Dir geht es $Antwort."
```



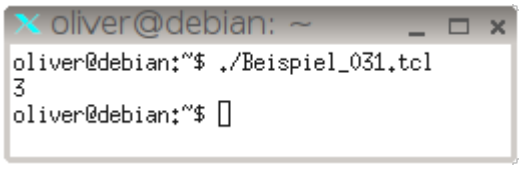
In Zeile 3 wird die Option `-nonewline` ergänzt. In Zeile 4 wird der Text durch den Befehl `flush stdout` angezeigt. Dieser Befehl ist notwendig, weil Tcl/Tk einen Text erst dann anzeigt, wenn der Text mit einem `newline`-Zeichen abgeschlossen wird. Da in Zeile 3 aber das `newline`-Zeichen unterdrückt wurde, muss man mit dem Befehl `flush stdout` selbst dafür sorgen, dass der Text angezeigt wird.

6.6 Eckige Klammern

Grundsätzlich wird in jede Zeile genau ein Befehl geschrieben. Wenn man aber den Rückgabewert eines Befehls als Argument für einen anderen Befehl braucht, schachtelt man die Befehle mit eckigen Klammern [ ] in einander.

Listing 6.21: Eckige Klammern (Beispiel031.tcl)

```
1 #!/usr/bin/env tclsh
2 puts [expr 1+2]
```



In Zeile 2 wird zuerst der Rechen-Befehl `expr` ausgewertet (der Befehl wird später beschrieben) und danach dessen Ergebnis an den Befehl `puts` übergeben.

6.7 Programm beenden

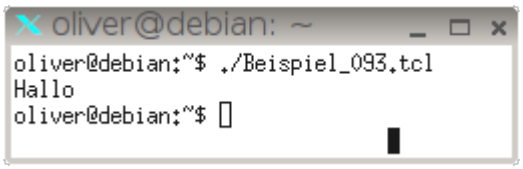
Befehl:

- `exit`

Üblicherweise wird der Programmcode von der ersten bis zur letzten Zeile ausgeführt. Damit endet das Programm. Mit dem Befehl `exit` kann man das Programm an jeder beliebigen Stelle beenden, auch innerhalb einer Prozedur.

Listing 6.22: Exit im Hauptteil des Programms (Beispiel093.tcl)

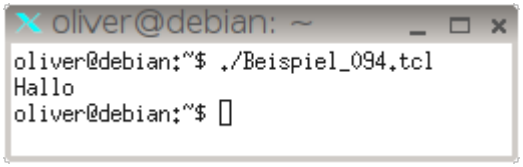
```
1 #!/usr/bin/env tclsh
2
3 puts "Hallo"
4 exit
5 puts "Diese Zeile wird nicht mehr ausgefuehrt."
```



Das nächste Beispiel greift dem Kapitel 9 auf Seite 69 vor, soll aber zeigen, dass man an jeder Stelle eines Programms den `exit`-Befehl verwenden kann.

Listing 6.23: Exit in einer Prozedur (Beispiel094.tcl)

```
1 #!/usr/bin/env tclsh
2
3 proc Ende {} {
4   exit
5 }
6
7 puts "Hallo"
8 Ende
9 puts "Diese Zeile wird nicht mehr ausgefuehrt."
```



In Zeile 8 wird die Prozedur `Ende` aufgerufen. In Zeile 4 wird das Programm sofort beendet und die Zeile 9 im Hauptteils des Programms wird nicht mehr ausgeführt.

6.8 Programm kommentieren

Kommentare helfen Ihnen und anderen, das Programm besser und schneller zu verstehen. Kommentare beginnen immer mit dem # Zeichen und werden entweder in eine eigene Zeile geschrieben oder (durch ein Semikolon abgetrennt) an das Zeilenende.

Listing 6.24: Kommentare (Beispiel064.tcl)

```
1 #!/usr/bin/env tclsh
```

```
2
3 # Zahlenraten:
4 # Das Programm erzeugt zufaellig eine Zahl zwischen 1 und 5.
5 # Der Spieler soll die Zahl erraten.
6
7 # Zufallszahl erzeugen
8 expr srand(int([clock seconds]))
9 set Geheimzahl [expr 1+int(5*rand())]
10
11 # Rateversuche des Spielers
12 set Loesung 0
13 while {$Loesung != $Geheimzahl} {
14   puts "Gib eine Zahl ein:"
15   set Loesung [gets stdin] ; # Tastatureingabe
16 }
17 puts "Du hast die Geheimzahl erraten. Die Geheimzahl war $Loesung."
```

Die Zeilen 3 bis 5 sind Kommentarzeilen, ebenso die Zeilen 7 und 11. In Zeile 15 wurde ein Kommentar am Zeilenende hinzugefügt. Dazu muss man hinter den Programmcode ein Semikolon setzen gefolgt vom Kommentarzeichen.

6.9 Programm mit Parametern starten

Befehle:

- `puts $argv0`
- `puts $argc`
- `puts $argv`
- `puts [lindex $argv 0]`

Man kann direkt beim Aufrufen des Programms Parameter übergeben. Diese werden in den Variablen `argv0`, `argc` und `argv` gespeichert.

Tabelle 6.1: Parameter

Variable	Beschreibung
`argv0`	Name des Skripts
`argc`	Anzahl der Argumente
`argv`	Liste mit allen Argumenten
`[lindex $argv 0]`	Erstes Argument
`[lindex $argv 1]`	Zweites Argument
usw.	usw.

Ein Beispiel zeigt die Parameterübergabe an das Programm.

Listing 6.25: Parameterübergabe (Beispiel326.tcl)

```
#!/usr/bin/env tclsh

puts "Skriptname: $argv0"
puts "Anzahl Argumente: $argc"
if {$argc > 0} {
  puts "Argumente: $argv"
  set Element [lindex $argv 0]
  puts "1. Argument: $Element"
}
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_326.tcl a b cde 123
Skriptname: ./Beispiel_326.tcl
Anzahl Argumente: 4
Argumente: a b cde 123
1. Argument: a
oliver@debian:~$
```

Wenn man das Programm mit den Argumenten `a b cde 123` startet, enthält die Variable `argv0` den Skriptnamen. Die Variable `argc` gibt an, dass vier Argumente übergeben wurden. Die Argumente sind als Liste in der Variablen `argv` gespeichert. Auf die einzelnen Elemente der Liste `argv` kann man mit Listenbefehlen zugreifen, wie zum Beispiel `lindex $argv 0`. Die Listenbefehle werden in Kapitel 11 auf Seite 97 beschrieben.

7 Zahlen, Datum und Uhrzeit

7.1 Zahlen

7.1.1 Rechnen mit Zahlen

Befehl:

- `[expr]`

Der Befehl zum Rechnen heißt `expr`. Üblicherweise wird der gesamte Befehl `expr` in eckige Klammern `[]` gesetzt. Es gibt folgende Operatoren:

Tabelle 7.1: Operatoren

Operator	Beschreibung
+	plus (addieren)
–	minus (subtrahieren)
*	mal (multiplizieren)
/	teilen (dividieren)
**	hoch (potenzieren)
%	Modulo, d. h. der Rest beim Teilen (z.B. 10

Zusätzlich zu den Grundrechenarten gibt es noch Funktionen wie z. B.

Tabelle 7.2: Funktionen

Funktion	Beschreibung
`sin(x)`	Sinus von x
`cos(x)`	Cosinus von x
`tan(x)`	Tangens von x
`asin(x)`	Arkussinus von x
`acos(x)`	Arkuscosinus von x
`atan(x)`	Arkustangens von x
`round(x)`	auf-/abrunden auf die nächste Ganzzahl
`ceil(x)`	aufrunden auf die nächste größere Ganzzahl
`floor(x)`	abrunden auf die nächste kleinere Ganzzahl
`max(x,y,z,...)`	Maximum aus x, y, z, ...
`min(x,y,z,...)`	Minimum aus x, y, z, ...
`rand()`	Zufallszahl im Bereich 0 bis 1 (inklusive 0, exklusive 1)
`pow(x,y)`	Exponentialrechnung x hoch y
`exp(x)`	Exponentialrechnung e hoch x
`sqrt(x)`	Quadratwurzel aus x
`log10(x)`	Logarithmus zur Basis 10
`log(x)`	Logarithmus zur Basis e
`fmod(x,y)`	Nachkommateil des Ergebnisses aus x geteilt durch y
`abs(x)`	Absolutwert der Zahl x
`int(x)`	Fließkommazahl x in Ganzzahl umwandeln
`double(x)`	Ganzzahl x in Fließkommazahl umwandeln

Es gilt, wie in der Mathematik üblich, die Punkt-vor-Strich-Rechnung. Runde Klammern werden ebenfalls beachtet. Die Winkelfunktionen (`sin()` usw.) sind zur Basis Pi (es gilt: Pi=180 Grad). Wenn das Rechenergebnis unendlich ist, wird der Text `Inf` (steht für infinity) zurückgegeben (bei minus unendlich entsprechend `-Inf`).

Da Tcl/Tk (wie auch andere Programmiersprachen) versucht, immer mit ganzzahligen

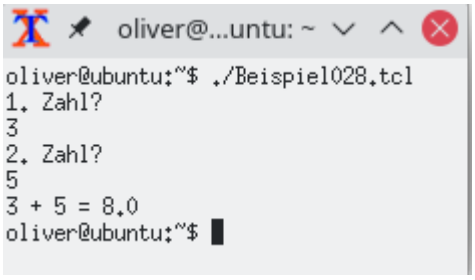
Zahlen zu rechnen, sollte man alle Variablen in die Funktion `double()` setzen und alle Konstanten (wenn sie keine Nachkommastellen haben) mit `.0` eingeben. Man schreibt somit statt `5 + $x` besser `5.0 + double($x)`.

Listing 7.1: Rechnen mit Konstanten (Beispiel027.tcl)

```
#!/usr/bin/env tclsh

set Ergebnis [expr 5.0-4.0]
puts "Das Ergebnis ist $Ergebnis."
```

Listing 7.2: Rechnen mit Variablen (Beispiel028.tcl)

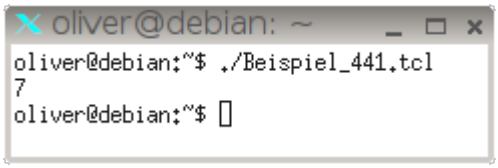
```
#!/usr/bin/env tclsh

puts "1. Zahl?"
set Zahl1 [gets stdin]
puts "2. Zahl?"
set Zahl2 [gets stdin]
set Ergebnis [expr double($Zahl1) + double($Zahl2)]
puts "$Zahl1 + $Zahl2 = $Ergebnis"
```



Man darf in Tcl/Tk auch die Operatoren durch eine Variable ersetzen.

Listing 7.3: Variable als Operator (Beispiel441.tcl)

```
#!/usr/bin/env tclsh

set Operator "+"
puts [expr 3.0 $Operator 4.0]
```



In Zeile 3 wird der Variable `Operator` ein Pluszeichen zugeordnet. In Zeile 4 wird beim Ausführen des Programms die Variable durch ein Pluszeichen ersetzt, so dass die Zeile wie folgt aussieht: `puts [expr 3 + 4]`.

Listing 7.4: Unendlich (Beispiel435.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Ergebnis [expr 1.0/0]
4 puts "1/0 = $Ergebnis"
5
6 set Ergebnis [expr -1.0/0]
7 puts "-1/0 = $Ergebnis"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_435.tcl
1/0 = Inf
-1/0 = -Inf
oliver@debian:~$
```

In Zeile 3 wird eine positive Zahl durch Null geteilt. Das Ergebnis ist unendlich und wird als `Inf` zurückgegeben. In Zeile 6 wird eine negative Zahl durch Null geteilt. Das Ergebnis ist minus unendlich, dargestellt durch `-Inf`.

Listing 7.5: Funktionen (Beispiel029.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Ergebnis [expr min(2.0,1.0,6.0)]
4 puts "Das Minimum aus den Zahlen 2, 1 und 6 ist $Ergebnis.
  "
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_029.tcl
Das Minimum aus den Zahlen 2, 1 und 6 ist 1.
oliver@debian:~$
```

Die trigonometrischen Funktionen `sin()`, `cos()` und `tan()` erwarten als Argument einen Radiant. Will man mit einer Grad-Angabe rechnen, so muss man folgende Umrechnung machen: Radiant = Grad*Pi/180, wobei Pi=3.141592653589793 ist.

Listing 7.6: Trigonometrische Funktionen (Beispiel306.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Pi 3.141592653589793
4 set Grad 30.0
5 set Sinus [expr sin(double($Grad)*$Pi/180.0)]
6 set Cosinus [expr cos(double($Grad)*$Pi/180.0)]
7 set Tangens [expr tan(double($Grad)*$Pi/180.0)]
```

```
8
9  puts "sin($Grad) = $Sinus"
10 puts "cos($Grad) = $Cosinus"
11 puts "tan($Grad) = $Tangens"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_306.tcl
sin(30) = 0.49999999999999994
cos(30) = 0.8660254037844387
tan(30) = 0.5773502691896257
oliver@debian:~$
```

7.1.2 Zahlen mit führender Null

Bei Tcl/Tk in den Versionen kleiner als 9.0 muss man aufpassen, wenn eine Ganzzahl eine führende Null hat, z. B. 012 statt 12. In diesem Fall wird die Zahl nicht als Dezimalzahl-Zahl, sondern als Oktal-Zahl interpriert. Oktalzahlen sind Zahlen mit der Basis 8. Die Zahl 012 wird somit als 1 x 8 + 2 = 10 interpretiert.

Listing 7.7: Ganzzahl mit führender Null ist eine Oktalzahl (Beispiel385.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts [expr int(010)]
4 puts [expr int(011)]
5 puts [expr int(012)]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_385.tcl
8
9
10
oliver@debian:~$
```

In den Zeilen 3 bis 5 werden die Zahlen 010, 011 und 012 als Oktalzahlen (d. h. Zahlen mit der Basis 8) interpretiert, so dass die Ausgabe nicht 10, 11 und 12 ist, sondern 8, 9 und 10.

7.1.3 Zufallszahlen

Befehle:

- `expr srand([clock seconds])`
- `set Zufallszahl [expr rand()]`

Mit dem Befehl `[expr srand([clock seconds])]` initialisiert man zuerst den Zufallszahlengenerator. Dies macht man einmalig beim Programmstart. Es wird die Anzahl

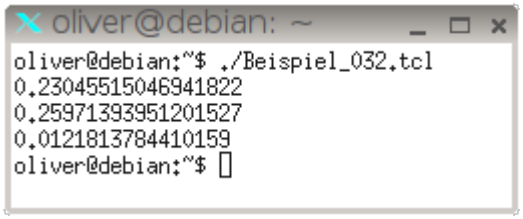
der Sekunden seit dem 1.1.1970 als Startwert genommen, so dass bei jedem Programmstart der Zufallszahlengenerator mit einem anderen Startwert startet.

Ein Computer erzeugt keine wirklichen Zufallszahlen, sondern es werden ausgehend von einem Startwert eine Folge von Zahlen erzeugt, die einer zufälligen Verteilung möglichst nahe kommen sollen. Dies bedeutet, dass der gleiche Startwert für den Zufallsgenerator immer zu derselben Abfolge an Zufallszahlen führt. Deshalb initialisiert man am Besten mit dem Befehl `[clock seconds]`.

Mit dem Befehl `[expr rand()]` erzeugt man eine Zufallszahl zwischen 0 und 1. Für die Zufallszahl gilt: 0 <= Zufallszahl < 1

Listing 7.8: Zufallszahlen (Beispiel032.tcl)

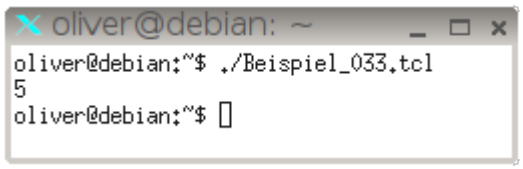
```
1 #!/usr/bin/env tclsh
2
3 expr srand([clock seconds])
4
5 puts [expr rand()]
6 puts [expr rand()]
7 puts [expr rand()]
```



In Zeile 3 wird der Zufallszahlengenerator initialisiert. In den Zeilen 5 bis 7 werden drei Zufallszahlen ausgegeben.

Listing 7.9: Zufallszahl im Bereich 0 bis 9 (Beispiel033.tcl)

```
1 #!/usr/bin/env tclsh
2
3 expr srand([clock seconds])
4 puts [expr int(10*rand())]
```



Listing 7.10: Zufallszahl im Bereich 1 bis 9 (Beispiel034.tcl)

```
1 #!/usr/bin/env tclsh
2
3 expr srand([clock seconds])
4 puts [expr 1+int(9*rand())]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_034.tcl
4
oliver@debian:~$
```

Listing 7.11: Zufällig 0 und 1 erzeugen (Beispiel035.tcl)

```
1 #!/usr/bin/env tclsh
2
3 expr srand([clock seconds])
4 puts [expr int(round(rand()))]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_035.tcl
0
oliver@debian:~$
```

7.1.4 Eine Variable erhöhen oder reduzieren

Befehl:

- `incr Variable Wert`

Der Befehl `incr` erhöht oder reduziert eine Variable, die eine ganzzahlige Zahl enthält, um einen ganzzahligen Wert. Wenn kein Wert angegeben wird, wird die Variable um 1 erhöht.

Listing 7.12: Variable erhöhen oder reduzieren (Beispiel036.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Zahl 0
4
5 incr Zahl
6 incr Zahl 8
7 incr Zahl -5
8
9 puts "Die Zahl ist $Zahl."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_036.tcl
Die Zahl ist 4.
oliver@debian:~$
```

In Zeile 5 wird die Variable `Zahl` um 1 erhöht, weil kein Wert hinter der Variablen angegeben wird. In Zeile 6 wird die Variable um 8 erhöht, in Zeile 7 um 5 reduziert.
Der Befehl `incr` funktioniert nur mit ganzzahligen Variablen, die man um einen ganzzahligen Wert verändern möchte. `incr` wird oft in `for`-Schleifen verwendet. Der Befehl `incr` ist schneller als ein mathematischer Ausdruck wie `set Variable [expr $Variable + 1]`.

7.2 Datum und Uhrzeit

Befehle:

- `puts [clock seconds]`
- `puts [clock clicks -milliseconds]`
- `puts [clock format [clock seconds] -format {%H:%M:%S}]`
- `puts [clock format [clock seconds] -format {%d.%m.%Y}]`
- `puts [clock format [clock seconds] -format {%d.%m.%Y %H:%M:%S}]`
- `puts [clock format [clock seconds] -format {%Y-%m-%d}]`
- `puts [clock format [clock seconds] -format {%d. %B %Y} -locale de]`
- `set Zeit [clock scan "$hours:$min:$sec $year-$mon-$mday"]`

Der Befehl `clock seconds` liefert die Anzahl der Sekunden seit dem 1.1.1970 0:00 Uhr UTC.

Der Befehl `clock clicks -milliseconds` misst die Zeit in Millisekunden. Dieser Befehl sollte nur zur Berechnung von Zeitdifferenzen verwendet werden, nicht zur Anzeige von Datum oder Uhrzeit.

Der Befehl `clock format` formatiert eine Zeitangabe. Die Format-Parameter setzt man am Besten in geschweifte Klammern `{}`.

Die Option `-locale de` bewirkt, dass die Tage und Monate in deutscher Sprache ausgegeben werden.

Der Befehl `clock scan` wird verwendet, wenn man eine Datum-Uhrzeit-Angabe in eine Variable speichern möchte.

7.2.1 Formatierung

Die wichtigsten Formatierungen sind:

Tabelle 7.3: Formatierungen

Formatierung	Beschreibung
`%H`	Stunden (0-23)
`%M`	Minuten (0-60)
`%S`	Sekunden (0-60)
`%d`	Tag im Monat (1-31)
`%j`	Tag im Jahr (immer dreistellig 001-366)
`%m`	Monat (1-12)
`%Y`	Jahr (4-stellig)
`%y`	Jahr (2-stellig)
`%b`	abgekürzter Monatsname (Jan, Feb, ...)
`%B`	voller Monatsname (Januar, Februar, ...)
`%a`	abgekürzter Name des Tages (Mon, Die, ...)
`%A`	voller Name des Tages (Montag, Dienstag, ...)
`%u`	Nummer des Tages innerhalb der Woche (1=Montag, 7=Sonntag)
`-locale de`	Ausgabe der Tage und Monate in deutsch

Listing 7.13: Datum und Zeit formatieren (Beispiel037.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts "Uhrzeit: [clock format [clock seconds] -format {%H:%M:%S}]"
4 puts "Datum: [clock format [clock seconds] -format {%d.%m.%Y}]"
5 puts "Datum: [clock format [clock seconds] -format {%d. %B %Y} -locale de]"
6 puts "Datum: [clock format [clock seconds] -format {%Y-%m-%d}]"
7 puts "Datum und Uhrzeit: [clock format [clock seconds] -format {%d.%m.%Y %H:%M:%S}]"
8 puts "Heute ist [clock format [clock seconds] -format %A -locale de]."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_037.tcl
Uhrzeit: 21:18:17
Datum: 31.05.2017
Datum: 31. Mai 2017
Datum: 2017-05-31
Datum und Uhrzeit: 31.05.2017 21:18:17
Heute ist Mittwoch.
oliver@debian:~$
```

In den Zeilen 5 und 8 wurde die Option `-locale de` ergänzt, damit der Monat bzw. Wochentag in deutscher Sprache ausgegeben wird.

Listing 7.14: Datum- und Zeitformat für das gesamte Programm festlegen (Beispiel038.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Zeitformat "%d.%m.%Y %H:%M:%S"
4 puts [clock format [clock seconds] -format $Zeitformat]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_038.tcl
05.07.2014 22:19:37
oliver@debian:~$
```

In Zeile 3 wurde ein Zeitformat definiert, das an allen Stellen des Programms verwendet werden soll. In Zeile 4 wird das aktuelle Datum im vorher definierten Zeitformat ausgegeben.

Listing 7.15: Uhrzeit und Datum in einer Variablen speichern (Beispiel039.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Zeitformat "%d.%m.%Y %H:%M:%S"
4 set Jetzt [clock seconds]
5 set Datum [clock scan "19:35:48 2013-07-26"]
6 puts "Datum und Uhrzeit: [clock format $Jetzt -format ↩
    $Zeitformat]"
7 puts "Datum und Uhrzeit: [clock format $Datum -format ↩
    $Zeitformat]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_039.tcl
Datum und Uhrzeit: 05.07.2014 22:21:13
Datum und Uhrzeit: 26.07.2013 19:35:48
oliver@debian:~$
```

In Zeile 3 wird das Zeitformat festgelegt.In Zeile 4 wird die aktuelle Zeit gespeichert. In Zeile 5 wird eine selbst festgelegte Datum-Zeit-Angabe eingelesen und gespeichert. In den Zeilen 6 und 7 erfolgt die Ausgabe der Zeitangaben unter Beachtung des Zeitformats.

Listing 7.16: Zeitdifferenz in Millisekunden messen (Beispiel301.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Startzeit [clock clicks -milliseconds]
4 after 1000
5 set Stoppzeit [clock clicks -milliseconds]
6 set Zeitdifferenz [expr $Stoppzeit - $Startzeit]
7 puts "Zeitdifferenz: $Zeitdifferenz Millisekunden"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_301.tcl
Zeitdifferenz: 1002 Millisekunden
oliver@debian:~$
```

In Zeile 3 wird die Startzeit in Millisekunden gespeichert. Der Befehl `after 1000` in Zeile 4 hält das Programm eine Sekunde (= 1.000 Millisekungen) an. In Zeile 5 wird die aktuelle Zeit gespeichert. In Zeile 6 wird die Zeitdifferenz in Millisekunden berechnet.

7.2.2 Rechnen mit Datum und Uhrzeit

Befehle:

- `puts [clock add $Zeit $Wert $Einheit]`

Man kann folgende Einheiten verwenden:

Tabelle 7.4: Einheiten

Einheit	Beschreibung
`seconds`	Sekunden
`minutes`	Minuten
`hours`	Stunden
`days`	Tage
`weeks`	Wochen
`months`	Monate
`years`	Jahre

Rechnen mit Datum und Uhrzeit kann sehr komplex sein. Man denke nur an die unterschiedlich langen Monate und Schaltjahre sowie die Umstellung zwischen Sommer- und Winterzeit. Durch Verwendung der passenden Einheit im Befehl `clock add` werden diese Effekte automatisch beim Rechnen beachtet.

Listing 7.17: Sommer-/Winterzeit (Beispiel329.tcl)

```
1 #!/usr/bin/env tclsh
```

```
2
3 set Zeitformat "%d.%m.%Y %H:%M:%S"
4 set Datum1 [clock scan "11:00:00 2015-03-28"]
5 puts "Datum: [clock format $Datum1 -format $Zeitformat]"
6 set Datum2 [clock add $Datum1 24 hours]
7 puts "Datum + 24 Stunden: [clock format $Datum2 -format ↩
    $Zeitformat]"
8 set Datum3 [clock add $Datum1 1 days]
9 puts "Datum + 1 Tag: [clock format $Datum3 -format ↩
    $Zeitformat]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_329.tcl
Datum: 28.03.2015 11:00:00
Datum + 24 Stunden: 29.03.2015 12:00:00
Datum + 1 Tag: 29.03.2015 11:00:00
oliver@debian:~$
```

In der Nacht vom 28.3.2015 auf den 29.3.2015 wurde die Uhr von Winter- auf Sommerzeit umgestellt. In Zeile 6 werden zur Uhrzeit 11 Uhr am 28.3.2015 24 Stunden addiert. Durch die Umstellung der Uhr ergibt sich am 29.3.2015 als neue Uhrzeit jedoch nicht 11 Uhr, sondern 12 Uhr. In Zeile 8 wird zur Uhrzeit 11 Uhr am 28.3.2015 ein Tag addiert. Das ergibt als neue Uhrzeit 11 Uhr am 29.3.2015.

Listing 7.18: Schaltjahr (Beispiel330.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Zeitformat "%d.%m.%Y"
4 set Datum1 [clock scan "11:00:00 2013-02-28"]
5 puts "Datum (kein Schaltjahr): [clock format $Datum1 ↩
    -format $Zeitformat]"
6 set Datum2 [clock add $Datum1 2 days]
7 puts "Datum + 2 Tage: [clock format $Datum2 -format ↩
    $Zeitformat]"
8 puts ""
9 set Datum1 [clock scan "11:00:00 2012-02-28"]
10 puts "Datum (Schaltjahr): [clock format $Datum1 -format ↩
    $Zeitformat]"
11 set Datum2 [clock add $Datum1 2 days]
12 puts "Datum + 2 Tage: [clock format $Datum2 -format ↩
    $Zeitformat]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_330.tcl
Datum (kein Schaltjahr): 28.02.2013
Datum + 2 Tage: 02.03.2013

Datum (Schaltjahr): 28.02.2012
Datum + 2 Tage: 01.03.2012
oliver@debian:~$
```

Wie man sieht wurde automatisch erkannt, dass es sich bei dem Jahr 2012 um ein Schaltjahr handelt.

8 Programmablauf steuern

8.1 if-Bedingung

Der Befehl `if` ist eine wenn-dann-Konstruktion, wie z. B. wenn die Zahl kleiner als 0 ist, dann multipliziere die Zahl mit -1.

Befehl 1: einfachste Form

```
if {Bedingung} {
 Anweisungen
}
```

Befehl 2: Form mit einer Alternative

```
if {Bedingung} {
 Anweisungen
} else {
 Anweisungen
}
```

Befehl 3: Form mit mehreren Alternativen

```
if {Bedingung} {
 Anweisungen
} elseif {Bedingung} {
 Anweisungen
} elseif {Bedingung} {
 Anweisungen
} else {
 Anweisungen
}
```

Der Befehl `if` prüft, ob eine Bedingung erfüllt ist und führt dann die dahinter stehende Anweisung aus. Mit `elseif` kann man weitere Bedingungen hinzufügen. Das ist optional. Mit `else` legt man fest, welche Anweisungen ausgeführt werden soll, wenn keine zuvor stehende Bedingung zutrifft. Auch das ist optional. Wenn eine Anweisung ausgeführt wurde, macht das Programm in der Zeile nach dem `if`-Befehl weiter. Es überspringt also die restlichen Teile des `if`-Befehls.

Man muss darauf achten, dass genau so viele Klammern geschlossen werden, wie geöffnet wurden. Man sollte deshalb, sobald man eine Klammer öffnet, sogleich die schließende Klammer setzen. Erst danach fügt man zwischen die beiden Klammern den Programmcode ein.

Die öffnende Klammer `{` muss sich immer am Zeilenende befinden, die schließende Klammer `}` muss sich immer am Zeilenanfang befinden.
Es gibt folgende Bedingungen:

Tabelle 8.1: Bedingungen

Bedingung	Beschreibung
`$Zahl1 == $Zahl2`	Zahl1 ist gleich Zahl2
`$Text1 eq $Text2`	Text1 ist gleich Text 2 (eq=equal)
`$Text1 == $Text2`	Text1 ist gleich Text2 (besser nicht verwenden)
`$Zahl1 != $Zahl2`	Zahl1 ist ungleich Zahl2
`$Text1 ne $Text2`	Text1 ist ungleich Text 2 (ne=not equal)
`$Text1 != $Text2`	Text1 ist ungleich Text2 (besser nicht verwenden)
`$Zahl1 > $Zahl2`	Zahl1 ist größer als Zahl2
`$Text1 > $Text2`	Text1 kommt ASCII-sortiert nach Text2
`$Zahl1 < $Zahl2`	Zahl1 ist kleiner als Zahl2
`$Text1 < $Text2`	Text1 kommt ASCII-sortiert vor Text2
`$Zahl1 >= $Zahl2`	Zahl1 ist größer als Zahl2 oder gleich Zahl2
`$Text1 >= $Text2`	Text1 kommt ASCII-sortiert nach Text2 oder ist gleich Text2
`$Zahl1 <= $Zahl2`	Zahl1 ist kleiner als Zahl2 oder gleich Zahl2
`$Text1 <= $Text2`	Text1 kommt ASCII-sortiert vor Text2 oder ist gleich Text2
`$Element in $Liste`	Element ist in Liste enthalten
`$Element ni $Liste`	Element ist nicht in Liste enthalten (ni= not in)

Es gibt zwei logische Verknüpfungen: `&&` für eine und-Verknüpfung und `||` für eine oder-Verknüpfung. Die beiden senkrechten Striche erzeugt man mit der Tastenkombination `Alt` und `<`.

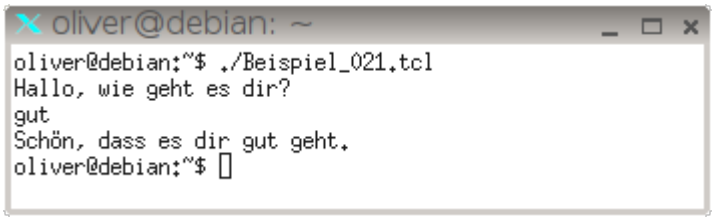
Tabelle 8.2: Logische Verknüpfungen

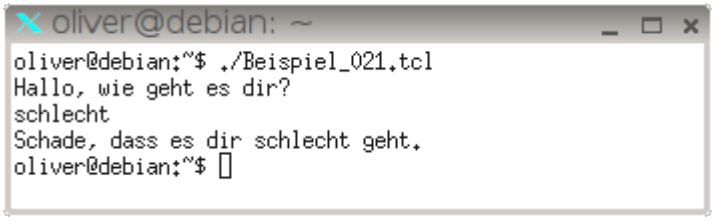
Logische Verknüpfung	Beschreibung
`$Zahl1 < 0 && $Zahl2 > 5`	Und-Verknüpfung
`$Zahl1 < 0 || $Zahl1 > 5`	Oder-Verknüpfung

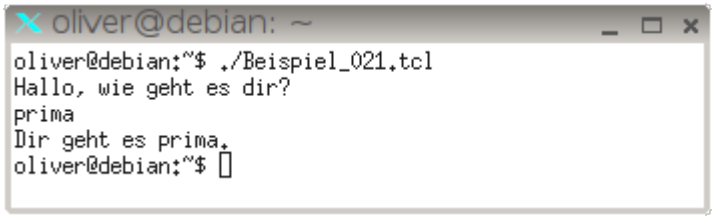
Listing 8.1: if-Bedingung (Beispiel021.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts "Hallo, wie geht es dir?"
4 set Antwort [gets stdin]
```

```
5  if {$Antwort == "gut"} {
6    puts "Schoen, dass es dir gut geht."
7  } elseif {$Antwort == "schlecht"} {
8    puts "Schade, dass es dir schlecht geht."
9  } else {
10   puts "Dir geht es $Antwort."
11 }
```







In Zeile 5 wird geprüft, ob in der Variablen `Antwort` das Wort `gut` gespeichert ist. Wenn ja, dann wird die Zeile 6 ausgeführt. Wenn nein, dann überspringt das Programm die Zeile 6 und macht in Zeile 7 weiter. Dort steht wieder eine Bedingung. Es wird geprüft, ob die Antwort `schlecht` lautet. Wenn ja, wird die Zeile 8 ausgeführt. Wenn nein, dann geht es in Zeile 9 weiter. Da alle bisherigen Bedingungen nicht zutrafen, wird die Zeile 10 ausgeführt.

Listing 8.2: Restliche Teile der if-Bedingung werden übersprungen (Beispiel442.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Zahl 5
4  if {$Zahl == 5} {
5    puts "1. Pruefung war erfolgreich."
6  } elseif {$Zahl < 10} {
7    puts "2. Pruefung war erfolgreich."
8  } else {
9    puts "3. Pruefung war erfolgreich."
10 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_442.tcl
1. Prüfung war erfolgreich.
oliver@debian:~$
```

In Zeile 3 wird der Wert 5 in die Variabel `Zahl` gespeichert. In Zeile 4 wird geprüft, ob die Zahl gleich 5 ist. Da das zutrifft, wird die Zeile 5 ausgeführt. Danach wird das Programm nach der `if`-Bedingung fortgeführt. Das heißt, dass alle weiteren Prüfungen der `if`-Bedingung in den Zeilen 6 und 8 nicht mehr beachtet werden.

Listing 8.3: Und-/ Oder-Bedingung (Beispiel022.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Zahl 5
4  if {$Zahl > 3 && $Zahl < 10} {
5    puts "Die Zahl ist groesser als 3 und kleiner als 10."
6  }
7
8  if {$Zahl == 5 || $Zahl == 10} {
9    puts "Die Zahl ist gleich 5 oder 10."
10 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_022.tcl
Die Zahl ist größer als 3 und kleiner als 10.
Die Zahl ist gleich 5 oder 10.
oliver@debian:~$
```

Listing 8.4: Die Bedingungen in bzw. ni (Beispiel023.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Farben {rot gelb gruen}
4  set Farbe1 gelb
5  set Farbe2 blau
6
7  if {$Farbe1 in $Farben} {
8    puts "$Farbe1 ist in der Liste enthalten."
9  }
10
11 if {$Farbe2 ni $Farben} {
12   puts "$Farbe2 ist nicht in der Liste enthalten."
13 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_023.tcl
gelb ist in der Liste enthalten.
blau ist nicht in der Liste enthalten.
oliver@debian:~$
```

Listing 8.5: Unterschied zwischen == und eq (bzw. != und ne) (Beispiel024.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Text1 "3"
4  set Text2 "03"
5
6  if {$Text1 == $Text2} {
7    puts "$Text1 ist gleich $Text2"
8  } else {
9    puts "$Text1 ist ungleich $Text2"
10 }
11
12 if {$Text1 eq $Text2} {
13   puts "$Text1 ist gleich $Text2"
14 } else {
15   puts "$Text1 ist ungleich $Text2"
16 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_024.tcl
3 ist gleich 03
3 ist ungleich 03
oliver@debian:~$
```

In der Zeile 6 wurde die Bedingung == verwendet. Hierbei wird der Text als Zahl interpretiert. Aus `03` wird `3`. Somit ist die Bedingung erfüllt, die beiden Texte sind (scheinbar) gleich. In der Zeile 12 wird die Bedingung `eq` verwendet. Jetzt wird der Vergleich in Textform durchgeführt mit dem Ergebnis, dass `3` ungleich `03` ist. Analog gilt das auch für die Befehle `!=` und `ne`. Man sollte deshalb bei Textvergleichen immer die Bedingungen `eq` bzw. `ne` verwenden und nur bei Zahlen die Bedingungen == bzw. `!=`.

8.2 switch-Befehl

Der Befehl `switch` ist eine Verzweigung, wie z. B.: Wenn die Ampel grün ist, fahre weiter. Wenn die Ampel gelb ist, bremse. Wenn die Ampel rot ist, halte an.

Befehl 1: Es gibt zu einer Bedingung eine einzeilige Anweisung

```
switch $Variable {
 Bedingung {Anweisung}
 Bedingung {Anweisung}
 Bedingung {Anweisung}
```

```
 ...
 default {Anweisung}
}
```

Befehl 2: Es gibt zu einer Bedingung mehrzeilige Anweisungen

```
switch $Variable {
 Bedingung {
 Anweisungen
 }
 Bedingung {
 Anweisungen
 }
 default {
 Anweisungen
 }
}
```

Befehl 3: Es gibt mehrere Bedingungen, die dieselbe Anweisung ausführen sollen

```
switch $Variable {
 Bedingung -
 Bedingung -
 Bedingung {Anweisung}
 default {Anweisung}
}
```

Der Befehl `switch` prüft verschiedene Bedingungen der Reihe nach durch. Sobald eine Bedingung erfüllt ist, wird die Anweisung ausgeführt. Man kann statt der Anweisung auch ein Minus-Zeichen schreiben. Dann wird die Anweisung der folgenden Bedingung ausgeführt. Der Befehl `switch` ist übersichtlicher als viele, in einander geschachtelte `if`-Bedingungen. Der `default`-Teil enthält die Anweisung, die ausgeführt wird, wenn alle anderen Bedingungen zuvor nicht erfüllt sind. Der `default`-Teil entspricht dem `else`-Teil im `if`-Befehl.

Listing 8.6: switch-Befehl (Beispiel025.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts "Hallo, wie geht es dir?"
4 set Antwort [gets stdin]
5 switch $Antwort {
6   gut {puts "Schoen, dass es dir gut geht."}
7   schlecht {puts "Schade, dass es dir schlecht geht."}
8   default {puts "Dir geht es $Antwort."}
9 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_025.tcl
Hallo, wie geht es dir?
gut
Schön, dass es dir gut geht.
oliver@debian:~$
```

Der switch-Befehl ist in diesem Beispiel übersichtlicher als geschachtelte `if`-Befehle.

Listing 8.7: Mehrere Bedingungen, dieselbe Anweisung (Beispiel026.tcl)

```
1  #!/usr/bin/env tclsh
2
3  puts "Eine Farbe:"
4  set Farbe [gets stdin]
5  switch $Farbe {
6    rot -
7    gelb -
8    gruen {puts "Die Farbe ist rot, gelb oder gruen."}
9    blau {puts "Die Farbe ist blau."}
10   default {puts "Die Farbe ist nicht rot, gelb, gruen, ↩
       blau."}
11 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_026.tcl
Eine Farbe:
rot
Die Farbe ist rot, gelb oder grün.
oliver@debian:~$
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_026.tcl
Eine Farbe:
gelb
Die Farbe ist rot, gelb oder grün.
oliver@debian:~$
```

In den Zeilen 6 bis 8 wird auf die Farben `rot`, `gelb`, `grün` geprüft. Falls die Farbe rot oder gelb oder grün ist, soll dieselbe Anweisung ausgeführt werden. Statt hinter jede Bedingung dieselbe Anweisung zu schreiben, kann man ein Minus-Zeichen setzen. Dadurch wird die nächste Anweisung (in diesem Fall von der Bedingung `grün`) ausgeführt.

8.3 while-Schleife

Befehl:

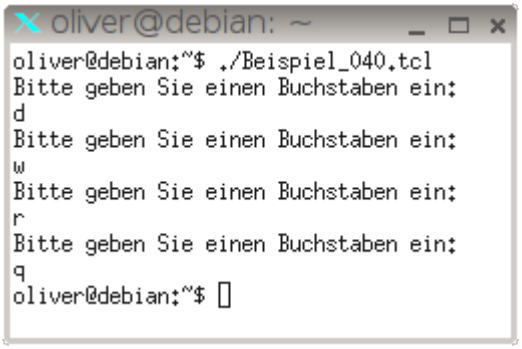
```
Initialisierung der Prüfvariable
while {Bedingung} {
 Anweisungen
 Verändern der Prüfvariable
}
```

Die `while`-Schleife ist eine Wiederholung von Befehlen, bis eine Abbruchbedingung erfüllt ist. In der Regel weiß man zu Beginn der Schleife nicht, wie oft die Schleife wiederholt wird. Zu Beginn der `while`-Schleife definiert man eine Variable, die in der Bedingung der `while`-Schleife überprüft wird. Dabei wird die Variable auf einen Anfangswert gesetzt, so dass die Bedingung beim ersten Mal erfüllt wird und die `while`-Schleife wenigstens

einmal durchlaufen wird. In der Schleife sollte die Variable, die in der Bedingung geprüft wird, verändert werden, so dass die `while`-Schleife irgendwann auch beendet wird.
Wenn man einen Fehler gemacht hat und die Schleife endlos ausgeführt wird, kann man das Programm mit `Strg+c` (`Strg`-Taste und `c`-Taste gleichzeitig drücken) beenden.

Listing 8.8: Die Schleife wird solange ausgeführt, bis die Taste q gedrückt wird (Beispiel040.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Eingabe ""
4 while {$Eingabe != "q"} {
5   puts "Bitte geben Sie einen Buchstaben ein:"
6   set Eingabe [gets stdin]
7 }
```



In Zeile 3 wird die Variable `Eingabe`, die in der `while`-Schleife als Abbruchkriterium verwendet wird, initialisiert. Dadurch ist sichergestellt, dass die Schleife wenigstens einmal durchlaufen wird. In Zeile 6 wird die Variable `Eingabe` verändert, in dem die Tastatureingabe in diese Variable gespeichert wird. Die `while`-Schleife läuft solange, wie die Bedingung in Zeile 4 erfüllt ist.

8.4 for-Schleife

Befehl:

```
for {Initialisierung} {Bedingung} {Veränderung}
{
 Anweisungen
}
```

Die `for`-Schleife ist eine Wiederholung von Befehlen, bis eine Abbruchbedingung erfüllt ist. In der Regel weiß man zu Beginn der Schleife, wie oft die Schleife durchlaufen werden soll.

Listing 8.9: Dreimal Hallo (Beispiel041.tcl)

```
1 #!/usr/bin/env tclsh
2
3 for {set i 1} {$i <=3} {incr i} {
```

```
4   puts Hallo
5 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_041.tcl
Hallo
Hallo
Hallo
oliver@debian:~$
```

In Zeile 3 wird in der ersten geschweiften Klammer `{set i 1}` der Startwert in der Variablen `i` gespeichert. In der zweiten geschweiften Klammer `{$i <=3}` wird die Bedingung definiert. Solange die Bedingung erfüllt ist, wird die `for`-Schleife ausgeführt. In der dritten geschweiften Klammer `{incr i}` wird die Variable verändert, in diesem Fall wird der Wert um 1 erhöht.

Listing 8.10: Summe der Zahlen 1 bis 100 (Beispiel042.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Summe 0
4 for {set Zahl 1} {$Zahl <=100} {incr Zahl} {
5   incr Summe $Zahl
6 }
7 puts "Die Summe der Zahlen 1 bis 100 ist $Summe."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_042.tcl
Die Summe der Zahlen 1 bis 100 ist 5050.
oliver@debian:~$
```

8.5 Schleifen vorzeitig beenden mit break und continue

Befehle:

- `break`
- `continue`

Es gibt zwei Befehle zum Beenden einer Schleife: `break` und `continue`. Mit dem `break`-Befehl beendet man die gesamte Schleife. Das Programm wird nach der Schleife fortgesetzt. Der `continue`-Befehl beendet nur den gerade ausgeführten Schleifendurchlauf und macht mit dem nächsten Durchlauf weiter. `Break` und `continue` können in allen Schleifen `for`, `while` und `foreach` (wird in Kapitel 11 auf Seite 97 behandelt) benutzt werden.

Listing 8.11: Schleife mit break beenden (Beispiel043.tcl)

```
1 #!/usr/bin/env tclsh
2
3 for {set Zahl 1} {$Zahl <=10} {incr Zahl} {
4   if {$Zahl == 5} {
5     break
6   }
7   puts $Zahl
8 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_043.tcl
1
2
3
4
oliver@debian:~$
```

In Zeile 4 wird geprüft, ob der Wert der Variablen `Zahl` gleich 5 ist. Wenn dies der Fall ist, wird der `break`-Befehl in Zeile 5 ausgeführt und damit die gesamte `for`-Schleife vorzeitig beendet.

Listing 8.12: Schleife mit continue beenden (Beispiel044.tcl)

```
1 #!/usr/bin/env tclsh
2
3 for {set Zahl 1} {$Zahl <=5} {incr Zahl} {
4   if {$Zahl == 3} {
5     continue
6   }
7   puts $Zahl
8 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_044.tcl
1
2
4
5
oliver@debian:~$
```

Wenn die Zahl gleich `3` ist (Zeile 4), wird der `continue`-Befehl in Zeile 5 ausgeführt. Damit wird die Ausführung der restlichen Schleife für die Zahl `3` beendet (d. h., die Zahl `3` wird nicht ausgegeben) und das Programm springt zurück in Zeile 3 und fährt mit der Zahl `4` fort.

9 Prozeduren

Eine Prozedur ist eine Zusammenfassung von Befehlen zu einem neuen Befehl.

Befehl 1: Einfache Prozedur

```
proc Name {} {
 Anweisungen
}
```

Befehl 2: Prozedur mit einem Parameter

```
proc Name {Variable} {
 Anweisungen
}
```

Befehl 3: Prozedur mit mehreren Parametern

```
proc Name {Variable1 Variable2} {
 Anweisungen
}
```

Befehl 4: Prozedur mit unbestimmter Anzahl Parameter

```
proc Name {Variable1 Variable2 args} {
 Anweisungen
}
```

Befehl 5: Prozedur mit Default-Wert

```
proc Name {Variable1 {Variable2 Default}} {
 Anweisungen
}
```

Befehl 6: Prozedur mit Rückgabewert

```
proc Name {Variable} {
 Anweisungen
 return Rückgabewert
}
```

Befehl 7: Prozedur mit Rückgabewert und ggf. vorzeitigem Ende

```
proc Name {Variable} {
 Anweisungen
 if {Bedingung} {
 return
 }
 return Rückgabewert
}
```

Befehl 8: Prozedur in Prozedur (sollte man nicht machen)

```
proc Name {Variable} {
 proc Name2 {} {
```

```
 Anweisungen
 }
 Anweisungen
}
```

9.1 Prozedur

Prozeduren fassen eine Abfolge von Befehlen unter einem neuen Namen zusammen. Man könnte deshalb auch sagen, dass man mit einer Prozedur einen neuen, eigenen Befehl erzeugt. Es gibt mindestens vier gute Gründe, Prozeduren zu verwenden:

- Wenn man Teile des Programms mehrfach ausführen möchte, braucht man die Befehle nur einmal in einer Prozedur abzulegen. Dann kann man mit einem einzigen Befehl (nämlich dem Starten der Prozedur) alle Befehle ausführen.
- Prozeduren machen das Programm leichter lesbar, wenn viele Befehle unter einem (sprechenden) Prozedurnamen zusammengefasst werden.
- Prozeduren erleichtern die Fehlersuche. Wenn die Prozedur einmal geprüft ist und richtig funktioniert, dann braucht man bei der Fehlersuche die Befehle in der Prozedur nicht mehr zu überprüfen.
- Man kann Prozeduren in verschiedenen Programmen verwenden, ohne jedes Mal die Prozedur neu zu programmieren.

Eine Prozedur beginnt mit dem Befehl `proc`. Danach folgt der Name der Prozedur und gegebenenfalls die an die Prozedur zu übergebenden Variablen. Die Variablen werden in geschweifte Klammern `{}` hinter den Prozedurnamen geschrieben. Wenn an die Prozedur keine Variablen übergeben werden, dann bleiben die geschweiften Klammern `{}` leer. Wenn man Werte an die Prozedur übergibt, werden die Werte der Reihe nach den Variablen im Prozedurkopf zugeordnet.

Arrays (siehe Kapitel 13 auf Seite 151) kann man auf diese Art nicht an die Prozedur übergeben. Hierzu muss man den Befehl `upvar` benutzen (siehe Kapitel 9.4 auf Seite 81).

Im Prozedurkopf kann man zu jeder Variable einen Default-Werte definieren. Die Variable bekommt den Default-Wert zugeordnet, wenn der Variablen beim Aufrufen der Prozedur kein Wert übergeben wird. Variablen mit Default-Wert werden in geschweifte Klammern `{}` gesetzt.

Wenn man eine unbestimmte Anzahl an Werten an die Prozedur übergeben möchte, schreibt man als letzte Variable `args`. Die Variable `args` nimmt alle restlichen Werte auf. Die Variable `args` ist eine Liste.

Die Variablen im Prozedurkopf müssen in folgender Reihenfolge angeordnet werden: zuerst die Variablen ohne Default-Wert, danach die Variablen mit Default-Wert, zuletzt `args`.

Mit dem Befehl `return` gibt man einen Wert zurück an den aufrufenden Programmteil (das ist optional). Will man mehrere Werte aus der Prozedur zurückgeben, speichert man die Werte als Liste und gibt die Liste zurück. Arrays kann man (auf diese Art) nicht zurückgeben. Hierfür muss man den Befehl `upvar` verwenden (siehe Kapitel 9.4 auf Seite 81).

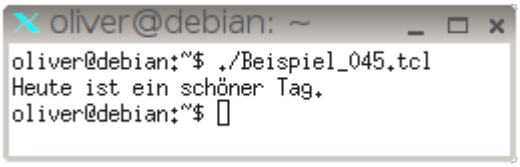
Mit dem Befehl `return` kann man eine Prozedur vorzeitig beenden.

Man kann innerhalb einer Prozedur eine andere Prozedur aufrufen. Dabei muss man beachten, dass die aufzurufende Prozedur vorher definiert wurde und somit dem Programm bereits bekannt ist.

Man kann innerhalb einer Prozedur auch eine weitere Prozedur definieren. Diese ist (im Unterschied zur vielleicht ersten Vermutung) nicht nur innerhalb der Prozedur gültig, sondern im gesamten Programm. Es kann somit zu Namenskonflikten mit anderen Prozeduren kommen, so dass man normalerweise darauf verzichten sollte, Prozeduren innerhalb einer anderen Prozedur zu definieren.

Listing 9.1: Prozedur ohne Übergabe von Werten (Beispiel045.tcl)

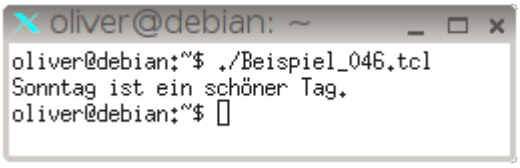
```
1 #!/usr/bin/env tclsh
2
3 proc SchoenerTag {} {
4   puts "Heute ist ein schoener Tag."
5 }
6
7 SchoenerTag
```



In den Zeilen 3 bis 5 wird die Prozedur definiert, aber noch nicht ausgeführt. Da die Prozedur keinen Wert übergeben bekommt, ist das erste Paar geschweifter Klammern `{}` leer. Nachdem die Prozedur definiert ist, kann sie im Hauptteil des Programms aufgerufen werden (Zeile 7).

Listing 9.2: Prozedur mit Übergabe eines Wertes (Beispiel046.tcl)

```
1 #!/usr/bin/env tclsh
2
3 proc SchoenerTag {Tag} {
4   puts "$Tag ist ein schoener Tag."
5 }
6
7 SchoenerTag Sonntag
```



In Zeile 3 wird festgelegt, dass die Prozedur eine Variable erwartet. Diese Variable wird dann innerhalb der Prozedur verwendet (Zeile 4). In Zeile 7 wird die Prozedur aufgerufen und dabei der Tag übergeben.

Listing 9.3: Prozedur mit Übergabe mehrerer Werte (Beispiel047.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc WieIstDerTag {Tag Eigenschaft} {
4    puts "$Tag ist ein $Eigenschaft Tag."
5  }
6
7  WieIstDerTag Montag schlechter
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_047.tcl
Montag ist ein schlechter Tag.
oliver@debian:~$
```

In Zeile 3 wird festgelegt, dass die Prozedur mehrere Variablen erwartet. Die Werte werden den Variablen von links nach rechts zugeordnet. In Zeile 7 wird die Prozedur aufgerufen. Der erste Wert ist `Montag` und wird in der Prozedur der ersten Variablen `Tag` zugeordnet. Der zweite Wert ist `schlechter`. Er wird der zweiten Variablen `Eigenschaft` zugeordnet.

Listing 9.4: Prozedur mit unbestimmter Anzahl an Werten (Beispiel048.tcl)

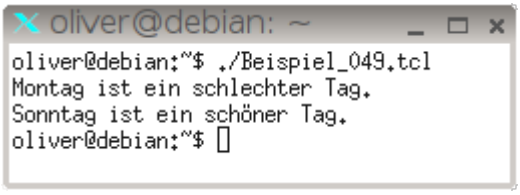
```
1  #!/usr/bin/env tclsh
2
3  proc SetzeSchrift {Art Groesse args} {
4    puts "Schriftart = $Art"
5    puts "Schriftgroesse = $Groesse"
6    puts "Schriftstile = $args"
7    puts "1. Schriftstil = [lindex $args 0]"
8    puts "2. Schriftstil = [lindex $args 1]"
9    puts "3. Schriftstil = [lindex $args 2]"
10   foreach Stil $args {
11     puts "Schriftstil = $Stil"
12   }
13 }
14
15 SetzeSchrift Helvetica 10 kursiv fett unterstrichen
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_048.tcl
Schriftart = Helvetica
Schriftgröße = 10
Schriftstile = kursiv fett unterstrichen
1. Schriftstil = kursiv
2. Schriftstil = fett
3. Schriftstil = unterstrichen
Schriftstil = kursiv
Schriftstil = fett
Schriftstil = unterstrichen
oliver@debian:~$
```

Die Prozedur erwartet drei Parameter: die Schriftart, die Schriftgröße und eine beliebige Anzahl an Schriftstilen. Diese werden in der Variablen `args` gespeichert (Zeile 3). Die Variable `args` ist eine Liste (siehe Kapitel 11 auf Seite 97). Die einzelnen Werte werden in den Zeilen 7 bis 9 bzw. in einer `foreach`-Schleife (Zeilen 10 bis 12) angezeigt.

Listing 9.5: Prozedur mit Default-Wert (Beispiel049.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc WieIstDerTag {Tag {Eigenschaft "schoener"}} {
4    puts "$Tag ist ein $Eigenschaft Tag."
5  }
6
7  WieIstDerTag Montag schlechter
8  WieIstDerTag Sonntag
```



In Zeile 3 wird festgelegt, dass die Prozedur mehrere Variablen erwartet. Sollte für die zweite Variable `Eigenschaft` kein Wert übergeben werden, bekommt die Variable den Wert `schöner` zugeordnet. Dies ist der Default-Wert. In Zeile 7 wird die Prozedur mit zwei Werten aufgerufen. Somit wird nicht der Default-Wert verwendet. In Zeile 8 wird die Prozedur nur mit einem Wert aufgerufen. Jetzt wird für die zweite Variable der Default-Wert genommen. Im Prozedurkopf (Zeile 3) müssen zuerst die Variablen ohne Default-Wert aufgelistet werden, danach die Variablen mit Default-Wert. Die Variablen mit Default-Wert müssen in geschweifte Klammern `{}` gesetzt werden.

Listing 9.6: Prozedur mit Default-Wert (Beispiel050.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc SetzeSchrift {Art {Groesse 10} {Stil {fett ↩
     unterstrichen}}} {
4    puts "Schriftart = $Art"
5    puts "Schriftgroesse = $Groesse"
6    puts "Schriftstile = $Stil"
7  }
8
9  SetzeSchrift Helvetica 14 kursiv
10 SetzeSchrift Courier 14 {kursiv durchgestrichen}
11 SetzeSchrift Times
12 SetzeSchrift Helvetica "" kursiv
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_050.tcl
Schriftart = Helvetica
Schriftgröße = 14
Schriftstile = kursiv
Schriftart = Courier
Schriftgröße = 14
Schriftstile = kursiv durchgestrichen
Schriftart = Times
Schriftgröße = 10
Schriftstile = fett unterstrichen
Schriftart = Helvetica
Schriftgröße =
Schriftstile = kursiv
oliver@debian:~$
```

In Zeile 3 wird festgelegt, dass zuerst die Schriftart übergeben wird, dann die Schriftgröße und als dritter Wert eine Liste mit Schriftstilen. Wenn keine Schriftgröße angegeben wird, wird die Größe `10` verwendet. Wenn kein Schriftstil übergeben wird, werden die Stile `fett` und `unterstrichen` genommen. In Zeile 9 wird die Prozedur mit einem Schriftstil aufgerufen. In Zeile 10 wird die Prozedur mit einer Liste an Schriftstilen aufgerufen. In Zeile 11 werden keine Angaben für die Schriftgröße und den -stil gemacht. Die Prozedur verwendet jetzt die Default-Werte. In Zeile 12 wird die Schriftgröße nur scheinbar weggelassen. Tatsächlich wurde aber ein leerer String an die Prozedur übergeben, so dass für die Schriftgröße nicht der Default-Wert genommen wurde. Das Beispiel zeigt, dass man keinen optionalen Parameter weglassen kann, wenn danach noch weitere Parameter folgen.

Listing 9.7: Prozedur mit einem Rückgabewert (Beispiel051.tcl)

```
1 #!/usr/bin/env tclsh
2
3 proc Quadrat {Zahl} {
4   set Ergebnis [expr $Zahl * $Zahl]
5   return $Ergebnis
6 }
7
8 puts [Quadrat 5]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_051.tcl
25
oliver@debian:~$
```

In Zeile 5 wird mit dem Befehl `return` ein Wert von der Prozedur an das Hauptprogramm zurückgegeben. In Zeile 8 wird zuerst die eckige Klammer ausgeführt, somit die Prozedur aufgerufen. Der Rückgabewert wird mit dem Befehl `puts` ausgegeben.

Listing 9.8: Prozedur mit mehreren Rückgabewerten (Beispiel052.tcl)

```
1 #!/usr/bin/env tclsh
2
3 proc Berechnungen {Zahl} {
```

```
4    set Quadrat [expr $Zahl * $Zahl]
5    set Wurzel [expr sqrt($Zahl)]
6    lappend Ergebnis $Quadrat $Wurzel
7    return $Ergebnis
8  }
9
10 set Ergebnis [Berechnungen 5]
11 lassign $Ergebnis Quadrat Wurzel
12 puts "Quadrat = $Quadrat"
13 puts "Wurzel = $Wurzel"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_052.tcl
Quadrat = 25
Wurzel = 2.23606797749979
oliver@debian:~$
```

Wenn man mehrere Werte aus der Prozedur an das Hauptprogramm zurückgeben will, muss man hilfsweise die Rückgabewerte zuerst mit dem Befehl `lappend` in einer Liste speichern (Zeile 6). Der Befehl `lappend` wird später genauer beschrieben. In Zeile 7 wird die Liste mit allen Rückgabewerten an das Hauptprogramm zurückgegeben. In Zeile 10 wird die Liste mit den Rückgabewerten in der Variablen Ergebnis gespeichert. In Zeile 11 werden die Rückgabewerte aus der Liste mit dem Befehl `lassign` extrahiert und verschiedenen Variablen zugeordnet. Der Befehl lassign wird später genauer beschrieben. In den Zeilen 12 und 13 werden die einzelnen Werte ausgegeben.

Listing 9.9: Prozedur vorzeitig verlassen (Beispiel053.tcl)

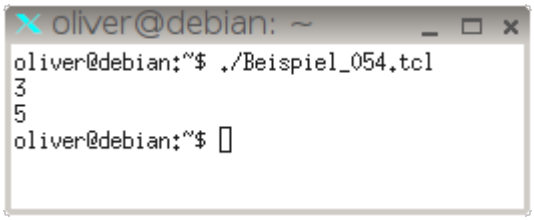
```
1  #!/usr/bin/env tclsh
2
3  proc Wurzel {Zahl} {
4    if {$Zahl < 0} {
5      puts "Bitte eine Zahl groesser Null eingeben!"
6      return
7    }
8
9    set Ergebnis [expr sqrt($Zahl)]
10   return $Ergebnis
11 }
12
13 puts "Wurzel aus -3 = [Wurzel -3]"
14 puts "Wurzel aus 4 = [Wurzel 4]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_053.tcl
Bitte eine Zahl größer Null eingeben!
Wurzel aus -3 =
Wurzel aus 4 = 2.0
oliver@debian:~$
```

In Zeile 6 wird die Prozedur vorzeitig beendet, wenn die Zahl kleiner 0 ist. Um die Prozedur übersichtlich zu gestalten, ist es ratsam, alle Bedingungen, die zu einem vorzeitigen Ende der Prozedur führen, direkt an den Anfang der Prozedur zu schreiben.

Listing 9.10: Eine Prozedur ruft eine andere Prozedur auf (Beispiel054.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc Absolutwert {Zahl} {
4    if {$Zahl < 0} {
5      set Zahl [VorzeichenUmkehren $Zahl]
6    }
7    return $Zahl
8  }
9
10 proc VorzeichenUmkehren {Zahl} {
11   return [expr -$Zahl]
12 }
13
14 puts [Absolutwert 3]
15 puts [Absolutwert -5]
```

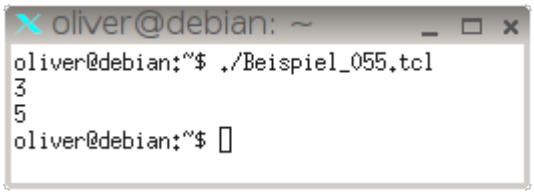


In den Zeilen 3 bis 8 wird die Prozedur `Absolutwert` definiert. Der Absolutwert einer Zahl ist immer die Zahl mit positivem Vorzeichen, also -5 wird zu +5. In den Zeilen 10 bis 12 wird die Prozedur `VorzeichenUmkehren` definiert. Das Hauptprogramm ruft in den Zeilen 14 und 15 die Prozedur `Absolutwert` auf. Die Prozedur `Absolutwert` prüft, ob die Zahl kleiner Null ist (Zeile 4) und ruft bei Bedarf die Prozedur `VorzeichenUmkehren` auf (Zeile 5). Die Prozedur `VorzeichenUmkehren` macht aus Minus-Zahlen Plus-Zahlen und umgekehrt. Die Zahl mit dem umgekehrten Vorzeichen wird in Zeile 11 an die aufrufende Prozedur `Absolutwert` zurückgegeben. Diese gibt in Zeile 7 die (neue) Zahl an das Hauptprogramm zurück, wo sie mit dem Befehl `puts` angezeigt wird.

An dem Beispiel erkennt man auch, dass die Reihenfolge, in der die Prozeduren definiert werden, nicht relevant ist. Erst wenn die Prozedur aufgerufen wird, muss sie dem Programm bekannt sein. Das sieht man daran, dass bereits in Zeile 5 die Prozedur `VorzeichenUmkehren` steht, aber diese Prozedur erst in Zeile 10 definiert wird. Es kommt zu keiner Fehlermeldung, weil der Tcl-Interpreter nach der Zeile 3 noch nicht in die Zeilen 4 bis 7 verzweigt (die Prozedur wird noch nicht ausgeführt), sondern in Zeile 10 weiter macht. Nach der Zeile 10 geht der Tcl-Interpreter in die Zeile 14. Dort erst wird die Prozedur `Absolutwert` aufgerufen, und erst jetzt gelangt der Tcl-Interpreter in die Zeilen 4 bis 7. Somit ist dem Tcl-Interpreter die Prozedur `VorzeichenUmkehren` bekannt, wenn er in die Zeile 5 gelangt. Der genaue Programmablauf wird in einem späteren Kapital noch im Detail erklärt.

Listing 9.11: Prozedur in einer Prozedur (Beispiel055.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc Absolutwert {Zahl} {
4    proc VorzeichenUmkehren {Zahl} {
5      return [expr -$Zahl]
6    }
7
8    if {$Zahl < 0} {
9      set Zahl [VorzeichenUmkehren $Zahl]
10   }
11   return $Zahl
12 }
13
14 puts [Absolutwert -3]
15 puts [VorzeichenUmkehren -5]
```



Die Prozedur `VorzeichenUmkehren` (Zeile 4) wird innerhalb der Prozedur `Absolutwert` (Zeile 3) definiert. Dennoch ist die Prozedur `VorzeichenUmkehren` auch vom Hauptprogramm aus aufrufbar. Es handelt sich demnach nicht um eine Prozedur, die nur innerhalb der Prozedur `Absolutwert` gültig ist, sondern um eine global gültige Prozedur. Dies ist ein wichtiger Unterschied zu lokalen Variablen (siehe Kapitel 9.3 auf Seite 78), die nur innerhalb der Prozedur gültig sind. Prozeduren sind immer global gültig, auch wenn sie innerhalb einer anderen Prozedur definiert wurden. Wenn Prozeduren nicht global gültig sein sollen, muss man Namensräume festlegen (siehe Kapitel 30 auf Seite 285). Um Namenskonflikte zu vermeiden, sollte man in Prozeduren keine weiteren Prozeduren definieren.

9.2 Prozedur auslagern

Befehl:

- `source Dateiname`

Man kann Prozeduren in separate Dateien auslagern. Das ist zum Beispiel bei Prozeduren sinnvoll, die man regelmäßig in verschiedenen Programmen benötigt. Mit dem Befehl `source` lädt man die Datei mit den Prozeduren in das Programm hinein. Erstellen Sie zunächst eine Datei mit folgender Prozedur:

Listing 9.12: Datei mit einer Prozedur (Beispiel336Prozeduren.tcl)

```
1 proc TextAnzeigen {Text} {
2   puts $Text
3 }
```

```
Prozedur.tcl
1  proc TextAnzeigen {Text} {
2     puts $Text
3  }
4
```

Die Datei speichern Sie unter dem Dateiname `Prozedur.tcl` in demselben Ordner ab, wie das nachfolgende Programm. Das eigentliche Programm sieht wie folgt aus:

Listing 9.13: Hauptprogramm (Beispiel336.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Skriptname [info script]
4 source [file join [file dirname $Skriptname] Prozedur.tcl]
5 TextAnzeigen "Guten Tag"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_336.tcl
Guten Tag
oliver@debian:~$
```

In Zeile 3 wird der Dateiname des Tcl-Programms ermittelt (mehr dazu in Kapitel 20.1 auf Seite 237). In Zeile 4 wird mit dem Befehl `source` die Datei `Prozedur.tcl` geladen. Dadurch steht die Prozedur `TextAnzeigen` im Programm zur Verfügung und wird in Zeile 5 aufgerufen.

9.3 Lokale und globale Variablen

Befehl 1: Prozedur global

```
proc Name {} {
 global Variable
 Anweisungen
}
```

Befehl 2: Prozedur mit zwei Doppelpunkten ::

```
proc Name {} {
 ::Variable
 Anweisungen
}
```

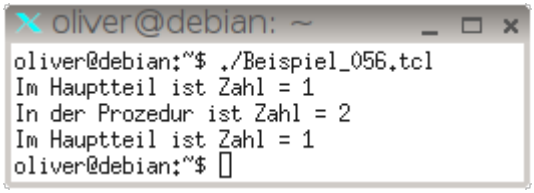
Variablen, die innerhalb einer Prozedur verwendet werden, sind zunächst nur lokal (d. h. nur innerhalb der Prozedur) gültig. Man kann mit ihnen nicht auf die Variablen außerhalb der Prozedur zugreifen. Erst der Befehl `global` innerhalb der Prozedur ermöglicht den Zugriff auf globale Variablen (das sind die Variablen aus dem Hauptteil des Programms). Alternativ kann man statt des Befehls `global` auch zwei Doppelpunkte `::` direkt vor die Variable schreiben.

Es ist empfehlenswert, nur eine (oder einige wenige) globale Variablen im Programm zu verwenden und die globale Variable als Array oder Dictionary anzulegen. Außerdem

sollte man der globalen Variablen einen Namen geben, der sofort erkennen lässt, dass es sich um eine globale Variable handelt, zum Beispiel `Glob`.

Listing 9.14: Die Variablen einer Prozedur sind immer lokal (Beispiel056.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc ZahlAnzeigen {} {
4    set Zahl 2
5    puts "In der Prozedur ist Zahl = $Zahl"
6  }
7
8  set Zahl 1
9  puts "Im Hauptteil ist Zahl = $Zahl"
10 ZahlAnzeigen
11 puts "Im Hauptteil ist Zahl = $Zahl"
```



Im Hauptteil des Programms wird in Zeile 8 eine Variable `Zahl` definiert und ihr der Wert `1` zugewiesen. In der Prozedur wird in Zeile 4 ebenfalls eine Variable `Zahl` definiert und darin der Wert `2` gespeichert. Die Variable aus der Prozedur hat jedoch nichts zu tun mit der Variablen aus dem Hauptteil, obwohl sie den gleichen Namen hat. Das sieht man daran, dass die Variable `Zahl` aus dem Hauptteil sowohl vor dem Aufrufen der Prozedur `ZahlAnzeigen` die Zahl `1` enthält (Zeile 9) als auch nach dem Aufrufen der Prozedur (Zeile 11).

Variablen gelten immer nur dort, wo sie definiert wurden. Man sagt dazu, dass es sich um lokale Variable handelt.

Listing 9.15: Der Befehl global (Beispiel057.tcl)

```
#!/usr/bin/env tclsh

proc ZahlAendern {} {
  global Zahl
  set Zahl 2
  puts "In der Prozedur ist Zahl = $Zahl"
}

set Zahl 1
puts "Im Hauptteil ist Zahl = $Zahl"
ZahlAendern
puts "Im Hauptteil ist Zahl = $Zahl"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_057.tcl
Im Hauptteil ist Zahl = 1
In der Prozedur ist Zahl = 2
Im Hauptteil ist Zahl = 2
oliver@debian:~$
```

In Zeile 4 wird die Variable `Zahl` als global definiert. Damit ist diese Variable nicht mehr lokal (also nur innerhalb der Prozedur) gültig, sondern es handelt sich um die Variable `Zahl` aus dem Hauptprogramm. In Zeile 9 wird der Variablen `Zahl` der Wert `1` zugeordnet. Durch die Prozedur `ZahlAendern` wird der Wert der Variablen `Zahl` auf `2` gesetzt. Somit hat die Variable `Zahl` im Hauptteil des Programms, nachdem die Prozedur `ZahlAendern` ausgeführt wurde, einen neuen Wert.

Listing 9.16: Globale Variable als Array anlegen (Beispiel303.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc GlobaleVariableAnzeigen {} {
4    global Glob
5
6    puts "Glob(Zahl) ist gleich $Glob(Zahl)"
7    puts "Glob(Text) ist $Glob(Text)"
8  }
9
10 set Glob(Zahl) 1
11 set Glob(Text) "Hallo"
12
13 GlobaleVariableAnzeigen
```

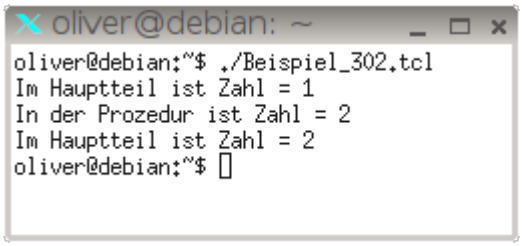
```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_303.tcl
Glob(Zahl) ist gleich 1
Glob(Text) ist Hallo
oliver@debian:~$
```

Es ist vorteilhaft, die globalen Variablen in einem Array oder Dictionary zusammenzufassen und der globalen Variablen einen Namen zu geben, der sofort erkennen lässt, dass es sich um eine globale Variable handelt. In dem Beispiel wird das Array `Glob` genannt. In den Zeilen 10 und 11 werden zwei Variablen in einem globalen Array angelegt. In der Prozedur in Zeile 4 wird das Array `Glob` als global definiert, so dass man auf die Variablen zugreifen kann. In den Zeilen 6 und 7 wird auf die einzelnen Variablen des Arrays zugegriffen.

Listing 9.17: Zwei Doppelpunkte :: (Beispiel302.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc ZahlAendern {} {
4    set ::Zahl 2
5    puts "In der Prozedur ist Zahl = $::Zahl"
```

```
6  }
7
8  set Zahl 1
9  puts "Im Hauptteil ist Zahl = $Zahl"
10 ZahlAendern
11 puts "Im Hauptteil ist Zahl = $Zahl"
```



In den Zeilen 4 und 5 werden direkt vor die Variable Zahl zwei Doppelpunkte `::` gesetzt. Dadurch wird die Variable als globale Variable erkannt. Genauer gesagt, definieren die zwei Doppelpunkte den globalen Namensraum (siehe Kapitel 30 auf Seite 285).

9.4 Upvar

Befehl:

```
proc Name {tmpVariable} {
 upvar $tmpVariable Variable
 Anweisungen
}
```

Wenn man in einer Prozedur eine Variable außerhalb der Prozedur verwenden will, kann man die Variable innerhalb der Prozedur global machen. Der Nachteil ist dabei, dass der Variablenname in der Prozedur geändert werden muss, wenn der Name im Hauptprogramm geändert wird. Es kann deshalb vorteilhaft sein, in der Prozedur statt des Befehls `global` den Befehl `upvar` zu verwenden, um auf die globale Variable zuzugreifen. Der Befehl `upvar` erzeugt einen Verweis auf eine globale Variable, so dass die Variablennamen innerhalb der Prozedur nicht mit den Variablennamen im restlichen Programm kollidieren.

Listing 9.18: Mit upvar auf globale Variablen zugreifen (Beispiel058.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc ZahlAendern {tmp} {
4    puts "Die Variable tmp hat folgenden Inhalt: $tmp"
5
6    upvar $tmp ZahlLokal
7
8    puts "Die Variable ZahlLokal ist ein Verweis auf $tmp ↩
       und hat folgenden Inhalt: $ZahlLokal"
9
10   set ZahlLokal 2
11   puts "Die Variable ZahlLokal wurde geaendert und hat ↩
       jetzt folgenden Inhalt: $ZahlLokal"
```

```
12    puts "Dadurch wurde aber auch die Variable ZahlGlobal ↩
         geaendert, wie man gleich sehen wird."
13 }
14
15 set ZahlGlobal 1
16 puts "Im Hauptteil ist ZahlGlobal gleich $ZahlGlobal"
17 ZahlAendern ZahlGlobal
18 puts "Die Variable ZahlGlobal wurde in der Prozedur ↩
      geaendert und ist jetzt gleich $ZahlGlobal."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_058.tcl
Im Hauptteil ist ZahlGlobal gleich 1
Die Variable tmp hat folgenden Inhalt: ZahlGlobal
Die Variable ZahlLokal ist ein Verweis auf ZahlGlobal und hat folgenden Inhalt: 1
Die Variable ZahlLokal wurde geändert und hat jetzt folgenden Inhalt: 2
Dadurch wurde aber auch die Variable ZahlGlobal geändert, wie man gleich sehen wird.
Die Variable ZahlGlobal wurde in der Prozedur geändert und ist jetzt gleich 2.
oliver@debian:~$
```

In Zeile 11 wird die Prozedur `ZahlAendern` aufgerufen. Dabei wird der Name der globalen Variablen übergeben. Es wird nicht der Wert, der in der Variablen `ZahlGlobal` gespeichert ist, übergeben, sondern nur das Wort `ZahlGlobal` (nicht `$ZahlGlobal`). In der Prozedur wird in Zeile 3 der Name der Variable (also das Wort `ZahlGlobal`) in der lokal gültigen Variablen `tmp` gespeichert. In Zeile 6 wird dann der Name der globalen Variablen der lokalen Variablen `ZahlLokal` zugeordnet. Mit dem Befehl `upvar` wird ein Verweis auf die globale Variable erzeugt, so dass die Prozedur auf die globale Variable sowohl lesend (Zeile 8) als auch schreibend (Zeile 10) zugreifen kann.

Bei `upvar` ist somit der Variablenname nur innerhalb der Prozedur gültig (so dass es keinen Namenskonflikt mit anderen Teilen des Programms gibt), aber der Inhalt der Variable ist dennoch global und kann in der Prozedur geändert werden.

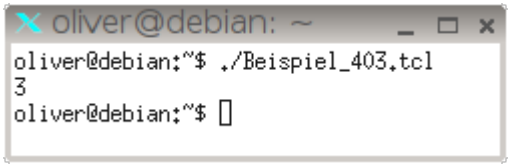
9.5 Uplevel

Befehle:
`uplevel`
`uplevel Ebene`
`uplevel #Ebene`

Der Befehl `uplevel` ermöglicht, Befehle in einem Programm-Umfeld außerhalb der Prozedur auszuführen. Wenn Sie beispielsweise im Hauptprogramm mehrere Befehle, die thematisch zusammengehören, durch einen einzigen Befehl zusammenfassen möchten, bietet es sich an, diese Befehle in eine Prozedur zu schreiben und die Ausführung der Prozedur mit `uplevel` in den Kontext des Hauptprogramms zu verschieben.

Listing 9.19: Uplevel führt Code eine Ebene höher im Hauptprogramm aus (Beispiel403.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc Rechnen {} {
4          uplevel {
5                  set c [expr $a + $b]
6          }
7  }
8
9  proc Ausgabe {} {
10         uplevel {
11                 puts $c
12         }
13 }
14
15 set a 1
16 set b 2
17 Rechnen
18 Ausgabe
```



In den Zeile 15 und 16 werden die beiden Variablen `a` und `b` definiert. In Zeile 17 wird die Prozedur `Rechnen` aufgerufen. Die Befehle in der Prozedur stehen stellvertretend für eine größere Anzahl thematisch zusammenhängender Befehle. Damit das Hauptprogramm besser lesbar wird, wurden diese Befehle (im Beispiel aus Platzgründen nur ein Befehl) in eine Prozedur ausgelagert. Üblicherweise werden die Befehle und Variablen innerhalb einer Prozedur lokal ausgeführt. Durch den `uplevel`-Befehl zu Beginn der Prozedur werden die Befehle aber jetzt im Kontext des Hauptprogramms ausgeführt. Das erkennt man daran, dass die Variablen `a` und `b` in der Prozedur bekannt sind und die Werte aus dem Hauptprogramm beinhalten. Auch die Ergebnis-Variable `c` ist im Hauptprogramm gültig, und kann deshalb in der Prozedur Ausgabe angezeigt werden.

Listing 9.20: Uplevel führt Code eine Ebene höher im Kontext einer Prozedur aus (Beispiel404.tcl)

```
#!/usr/bin/env tclsh

proc Rechnen {} {
        uplevel {
                set c [expr $a + $b]
        }
}

proc Ausgabe {} {
        uplevel {
                puts $c
        }
```

```
13 }
14
15 proc Start {} {
16         set a 1
17         set b 2
18         Rechnen
19         Ausgabe
20 }
21
22 Start
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_404.tcl
3
oliver@debian:~$
```

In Zeile 22 ruft das Hauptprogramm die Prozedur `Start` auf. In der Prozedur `Start` werden die beiden Variablen `a` und `b` definiert und dann die Prozeduren `Rechnen` und `Ausgabe` aufgerufen. Durch die `uplevel`-Befehle in den beiden Prozeduren werden die Befehle innerhalb der Prozeduren im Kontext der Prozedur `Start` ausgeführt.

Listing 9.21: Uplevel führt Code mehrere Ebene höher aus (Beispiel405.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc Rechnen {} {
4          uplevel 2 {
5                  set c [expr $a + $b]
6          }
7  }
8
9  proc Ausgabe {} {
10         uplevel 2 {
11                 puts $c
12         }
13 }
14
15 proc Start {} {
16         Rechnen
17         Ausgabe
18 }
19
20 set a 1
21 set b 2
22 Start
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_405.tcl
3
oliver@debian:~$
```

In den Zeilen 20 und 21 werden die beiden Variablen `a` und `b` definiert. Sie gelten im Hauptprogramm. In Zeile 22 wird die Prozedur `Start` aufgerufen, die ihrerseits die Prozeduren `Rechnen` und `Ausgabe` aufruft. Diese beiden Prozeduren sind somit zwei Ebenen unterhalb des Hauptprogramms. Durch den Befehl `uplevel 2` werden die beiden Prozeduren aber zwei Ebenen höher (also im Kontext des Hauptprogramms) ausgeführt.

Bislang war die `uplevel`-Angabe immer relativ zur aktuellen Ebene. Man kann aber auch eine feste Ebene angeben, in der der Code ausgeführt werden soll. Dazu schreibt man ein #-Zeichen direkt vor die Ebene.

Listing 9.22: Uplevel führt Code im Hauptprogramm aus (absolute Angabe der Ebene) (Beispiel460.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc Rechnen {} {
4          uplevel #0 {
5                  set c [expr $a + $b]
6          }
7  }
8
9  proc Ausgabe {} {
10         uplevel {
11                 puts $c
12         }
13 }
14
15 set a 1
16 set b 2
17 Rechnen
18 Ausgabe
```

```
oli@debian: ~
oli@debian:~$ ./Beispiel460.tcl
3
oli@debian:~$
```

In Zeile 4 wurde der `uplevel`-Befehl um ein #-Zeichen ergänzt. Dadurch gibt man fest an, in welcher Ebene der Code ausgeführt werden soll.

9.6 Trace

Befehle:
`trace add variable Variable Aktion Prozedur`

Tabelle 9.1: Aktionen

Aktion	Beschreibung
`write`	wenn der Inhalt der Variablen geändert wird
`read`	wenn der Inhalt der Variablen gelesen wird
`unset`	wenn die Variablen gelöscht wird

Der Befehl `trace` ermöglicht, auf Änderungen an bestimmten Variablen zu reagieren. Man kann damit zum Beispiel festlegen, dass immer eine bestimmte Prozedur ausgeführt wird, wenn sich der Inhalt einer Variablen ändert.

Listing 9.23: Trace reagiert, wenn eine Variable geändert wird (Beispiel567.tcl)

```
1  #!/usr/bin/tclsh
2
3  trace add variable Farbe write NeueFarbe
4
5  proc NeueFarbe {tmpName Element Aktion} {
6          upvar 1 $tmpName Name
7          puts "Bei der Variable $tmpName gab es folgende ↩
               Aktion: $Aktion"
8          puts "Die Variable $tmpName hat jetzt den Wert ↩
               $Name"
9  }
10
11 set Farbe "rot"
12 puts "---"
13 set Farbe "gelb"
```

```
oliver@pc:~$ ./Beispiel567.tcl
Bei der Variable Farbe gab es folgende Aktion: write
Die Variable Farbe hat jetzt den Wert rot
---
Bei der Variable Farbe gab es folgende Aktion: write
Die Variable Farbe hat jetzt den Wert gelb
oliver@pc:~$
```

In Zeile 3 wird der `trace`-Befehl definiert. Es wird für die Variable `Farbe` festgelegt, dass die Prozedur `NeueFarbe` ausgeführt wird, wenn die Variable geändert wird. In den Zeilen 5 bis 9 sieht man die Prozedur. In Zeile 5 werden drei Parameter an die Prozedur übergeben: der Variablenname, das Element der Variable (nur bei Array-Variablen) und die Aktion. In Zeile 6 wird eine Referenz auf die Variable im aufrufenden Programmteil (also dort, wo die Variable geändert wurde) erstellt (siehe Kapitel 9.4 auf Seite 81). In den Zeilen 7 und 8 erfolgt die Ausgabe.

Listing 9.24: Trace reagiert auf verschiedene Aktionen (Beispiel568.tcl)

```
1  #!/usr/bin/tclsh
2
3  trace add variable Farbe {write read unset} NeueFarbe
4
5  proc NeueFarbe {tmpName Element Aktion} {
6          upvar 1 $tmpName Name
7          puts "Bei der Variable $tmpName gab es folgende ↩
               Aktion: $Aktion"
8          switch $Aktion {
9                  "read" {puts "Es wurde der Wert $Name von ↩
                      der Variable $tmpName gelesen"}
10                 "write" {puts "Die Variable $tmpName hat ↩
                      jetzt den Wert $Name"}
11                 "unset" {puts "Die Variable $tmpName wurde↩
                       geloescht"}
12         }
13
14 }
15
16 set Farbe "rot"
17 puts "---"
18 puts "Hauptprogramm: $Farbe"
19 puts "---"
20 set Farbe "gelb"
21 puts "---"
22 puts "Hauptprogramm: $Farbe"
23 puts "---"
24 unset Farbe
```

```
oliver@pc:~$ ./Beispiel568.tcl
Bei der Variable Farbe gab es folgende Aktion: write
Die Variable Farbe hat jetzt den Wert rot
---
Bei der Variable Farbe gab es folgende Aktion: read
Es wurde der Wert rot von der Variable Farbe gelesen
Hauptprogramm: rot
---
Bei der Variable Farbe gab es folgende Aktion: write
Die Variable Farbe hat jetzt den Wert gelb
---
Bei der Variable Farbe gab es folgende Aktion: read
Es wurde der Wert gelb von der Variable Farbe gelesen
Hauptprogramm: gelb
---
Bei der Variable Farbe gab es folgende Aktion: unset
Die Variable Farbe wurde geloescht
oliver@pc:~$
```

In Zeile 3 wird festgelegt, dass die Prozedur `NeueFarbe` bei allen drei Aktionen `read`, `write` und `unset` ausgeführt werden soll.

Listing 9.25: Trace für mehrere Variablen (Beispiel569.tcl)

```
1  #!/usr/bin/tclsh
2
3  trace add variable Farbe1 write NeueFarbe
4  trace add variable Farbe2 write NeueFarbe
5
6  proc NeueFarbe {tmpName Element Aktion} {
7          upvar 1 $tmpName Name
8          puts "Bei der Variable $tmpName gab es folgende ↩
               Aktion: $Aktion"
9          puts "Die Variable $tmpName hat jetzt den Wert ↩
               $Name"
10 }
11
12 set Farbe1 "rot"
13 puts "---"
14 set Farbe2 "gelb"
```

```
oliver@pc:~$ ./Beispiel569.tcl
Bei der Variable Farbe1 gab es folgende Aktion: write
Die Variable Farbe1 hat jetzt den Wert rot
---
Bei der Variable Farbe2 gab es folgende Aktion: write
Die Variable Farbe2 hat jetzt den Wert gelb
oliver@pc:~$
```

In den Zeilen 3 und 4 werden zwei Variablen mittels `trace` überwacht.

Listing 9.26: Trace für eine Array-Variable (Beispiel570.tcl)

```
#!/usr/bin/tclsh

trace add variable Farbe write NeueFarbe

proc NeueFarbe {tmpName Element Aktion} {
        upvar 1 $tmpName Name
        puts "Bei der Variable ${tmpName}($Element) gab es↩
            folgende Aktion: $Aktion"
        puts "Die Variable ${tmpName}($Element) hat jetzt ↩
           den Wert $Name($Element)"
}

array set Farbe {}
set Farbe(1) "rot"
puts "---"
set Farbe(2) "gelb"
```

```
oliver@pc:~$ ./Beispiel570.tcl
Bei der Variable Farbe(1) gab es folgende Aktion: write
Die Variable Farbe(1) hat jetzt den Wert rot
---
Bei der Variable Farbe(2) gab es folgende Aktion: write
Die Variable Farbe(2) hat jetzt den Wert gelb
oliver@pc:~$
```

In den Zeilen 3 wird festgelegt, dass die Variable `Farbe` überwacht wird. In Zeile 11 wird die Variable als Array definiert. Wenn die Variable geändert wird und dadurch die Prozedur `NeueFarbe` ausgeführt wird, wird der Index der Array-Variable an den Parameter `Element` übergeben.

Das nächste Beispiel zeigt eine weitere Umsetzung der Trace-Prozedur, indem keine Parameter an die Prozedur übergeben werden.

Listing 9.27: Trace für mehrere Variablen (ohne Parameterübergabe) (Beispiel571.tcl)

```
1  #!/usr/bin/tclsh
2
3  set Zahl1 0
4  set Zahl2 0
5  set Ergebnis 0
6
7  trace add variable Zahl1 write Rechnen
8  trace add variable Zahl2 write Rechnen
9
10 proc Rechnen {- - -} {
11         global Zahl1
12         global Zahl2
13         global Ergebnis
14
15         set Ergebnis [expr double($Zahl1) + double($Zahl2)
           ]
16 }
17
18 puts "Zahl1=$Zahl1 Zahl2=$Zahl2 Ergebnis=$Ergebnis"
19 set Zahl1 5
20 puts "Zahl1=$Zahl1 Zahl2=$Zahl2 Ergebnis=$Ergebnis"
21 set Zahl2 3
22 puts "Zahl1=$Zahl1 Zahl2=$Zahl2 Ergebnis=$Ergebnis"
```

```
oliver@pc:~$ ./Beispiel571.tcl
Zahl1=0 Zahl2=0 Ergebnis=0
Zahl1=5 Zahl2=0 Ergebnis=5.0
Zahl1=5 Zahl2=3 Ergebnis=8.0
oliver@pc:~$
```

In Zeile 10 werden die Parameter der Prozedur `Rechnen` durch Minus-Zeichen ersetzt. In den Zeilen 11 bis 13 werden die relevanten Variablen als global definiert.

10 Programmablauf

10.1 Kurzform

Der tcl-Interpreter durchläuft das Programm genau einmal von oben nach unten und innerhalb einer Programmzeile von links nach rechts. Stößt er auf eine Prozedur, so merkt er sich den Namen der Prozedur, durchläuft aber noch nicht die Prozedur. Erst wenn die Prozedur aufgerufen wird, wertet der Interpreter den Inhalt der Prozedur aus. Deshalb zeigt der Interpreter auch solange keine Fehler innerhalb der Prozedur an, bis er die Prozedur ausführt.

Wenn ein Befehl einen anderen Befehl in eckigen Klammern enthält, wird zuerst der innere Befehl ausgewertet und dann das Ergebnis in den äußeren Befehl eingesetzt. Danach wird der äußere Befehl ausgewertet. Ein Beispiel:

```
set x [expr [set a 1] + 2 + $a]
```

Der tcl-Interpreter wertet die Programmzeile von links nach rechts aus. Er soll der Variablen `x` einen Wert zuordnen. Dabei stößt er auf die erste eckige Klammer. Innerhalb der eckigen Klammer steht ein neuer Befehl:

```
expr [set a 1] + 2 + $a
```

Dieser Befehl wird wieder von links nach rechts ausgewertet. Der Befehl `expr` besagt, dass etwas gerechnet werden soll. Dabei findet der tcl-Interpreter wieder eine eckige Klammer, die er zuerst auswertet:

```
set a 1
```

Der Interpreter weist der Variable `a` den Wert `1` zu. Zugleich ist der Rückgabewert des Befehls `set a 1` die Zahl `1`. Somit sieht der `expr`-Befehl jetzt wie folgt aus:

```
expr 1 + 2 + $a
```

Bei der weiteren Ausführung von links nach rechts findet der Interpreter die Variable `$a`. Da die Variable bereits mit der Zahl `1` belegt ist, kann der Interpreter die Variable durch die Zahl `1` ersetzen. Der Befehl sieht dann so aus:

```
expr 1 + 2 + 1
```

Nun kann der Interpreter das Ergebnis `4` ausrechnen. Das Ergebnis setzt er nun in den Befehl von ganz oben ein:

```
set x 4
```

Zum Schluss weist er der Variablen `x` den Wert `4` zu.

Bitte beachten Sie: Da der Interpreter die Befehle von links nach rechts auswertet, muss erst die Variable `a` definiert werden bevor sie benutzt werden kann. Der folgenden Befehl führt deshalb auch zu einem Fehler:

```
set x [expr $a + 2 + [set a 1]]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_059.tcl
can't read "a": no such variable
    while executing
"expr $a + 2 + [set a 1]"
    invoked from within
"set x [expr $a + 2 + [set a 1]]"
    (file "./Beispiel_059.tcl" line 2)
oliver@debian:~$
```

Der Interpreter versucht den Wert für die Variable `a` einzusetzen, bevor die Variable definiert wurde und einen Wert bekommen hat.
Noch ein Beispiel zum Programmablauf.

Listing 10.1: Programmablauf (Beispiel406.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc Eins {} {
4          puts "Prozedur Eins"
5          Zwei
6  }
7
8  proc Zwei {} {
9          puts "Prozedur Zwei"
10         exit
11 }
12
13 Eins
14 puts $Zahl
15 puts "Wird nicht angezeigt."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_406.tcl
Prozedur Eins
Prozedur Zwei
oliver@debian:~$
```

Der Interpreter startet in Zeile 1. In Zeile 3 trifft er auf die Prozedur `Eins`. Ab jetzt ist ihm der Befehl `Eins` bekannt. Der Interpreter wertet aber die Prozedur `Eins` nicht aus, weil die Prozedur noch nicht aufgerufen wird. Das erkennt man daran, dass es in Zeile 5 zu keiner Fehlermeldung kommt, obwohl dort der Befehl `Zwei` aufgerufen wird (also die Prozedur `Zwei`), die dem Interpreter zu diesem Zeitpunkt aber noch nicht bekannt wäre. Stattdessen überspringt der Interpreter den Inhalt der Prozedur `Eins` und trifft in Zeile 8 auf die Prozedur `Zwei`. Jetzt ist ihm auch der Befehl `Zwei` bekannt. Der Inhalt der Prozedur `Zwei` wird wieder übersprungen. In Zeile 13 trifft der Interpreter auf den Befehl `Eins`, der inzwischen bekannt ist. Er springt in die Zeile 4 und danach in Zeile 5. Dort steht der Befehl `Zwei`, der dem Interpreter mittlerweile ebenfalls bekannt ist. Er springt daraufhin in Zeile 9 und anschließend in Zeile 10. Dort wird das Programm beendet, so dass der Interpreter nicht mehr zur Zeile 14 springt. Das erkennt man daran, dass Zeile

14 eine Fehlermeldung verursachen würde, weil die Variable `Zahl` dem Interpreter nicht bekannt wäre. Auch die Zeile 15 wird nicht mehr ausgeführt. Die Reihenfolge, in der der Interpreter die Zeilen ausführt, ist somit: 3, 8, 13, 4, 5, 9, 10.

10.2 Details

Der Tcl-Interpreter wertet das Skript nach folgenden Regeln aus:

(1) Befehle

Ein Tcl-Skript ist ein Textdokument, das Befehle enthält. Semikolons und Neue-Zeile-Zeichen trennen die Befehle (Ausnahme: Semikolon und Neue-Zeile-Zeichen sind in Anführungszeichen gesetzt.)

(2) Auswertung der Befehle

Die Befehle werden in zwei Schritten ausgewertet. Zuerst teilt der Tcl-Interpreter den Befehl in seine einzelnen Bestandteile (= Wörter) auf und führt ggf. Ersetzungen (siehe unten) durch. Das erste Wort wird benutzt, um eine Prozedur zu finden, die den Befehl ausführen kann (deshalb muss man zum Rechnen den Befehl expr voranstellen). Anschließend werden alle weiteren Wörter des Befehls an diese Prozedur übergeben. Jede Prozedur interpretiert auf ihre eigene Art die einzelnen Wörter.

(3) Bestandteile eines Befehls

Die Bestandteile eines Befehls (= Wörter) werden mit einem Leerzeichen von einander getrennt. Die Bestandteile werden im Folgenden „Wörter“ genannt, der einzelne Bestandteil „Wort“.

(4) Doppelte Anführungszeichen `" "`

Wenn das erste Zeichen eines Wortes ein doppeltes Anführungszeichen `"` ist, geht das Wort bis zum nächsten doppelten Anführungszeichen `"`. Wenn innerhalb der Anführungszeichen ein Semikolon oder Leerzeichen steht, wird es als normales Zeichen ohne besondere Bedeutung behandelt. Befehlsersetzungen, Variablenersetzungen und Backslash-Ersetzungen werden (wie weiter unten beschrieben) innerhalb der Anführungszeichen durchgeführt. Die doppelten Anführungszeichen sind nicht Teil des Wortes.

(5) Expansion von Argumenten *

Wenn ein Wort mit * beginnt und das Zeichen direkt dahinter kein Leerzeichen ist, wird * entfernt und das Wort dahinter ganz normal vom Tcl-Interpreter interpretiert. Die Rückgabe aus dieser Wort-Interpretation erfolgt als Liste, deren einzelne Elemente als Wörter dem Befehl hinzugefügt werden. Beispiel:

```
set Liste [Datei1.txt Datei2.txt Datei3.txt]
file copy {*}$Liste Ordner
```

wird zu:

```
file copy Datei1.txt Datei2.txt Datei3.txt Ordner
```

(6) Geschweifte Klammern `{ }`

Wenn das erste Zeichen eines Wortes eine geschweifte Klammer `{` ist und Regel 5 nicht zutrifft, geht das Wort bis zur zugehörigen, schließenden Klammer `}`. Wenn geschweifte

Klammern in einander geschachtelt werden, müssen genau so viele schließende Klammern gesetzt werden wie geöffnet wurden. Innerhalb der geschweiften Klammern erfolgen keine Befehlsersetzungen und keine Variablenersetzungen. Nur die Backslash-Ersetzung wird ausgeführt. Leerzeichen und Semikolons haben keine besondere Bedeutung und werden als normale Zeichen interpretiert. Die geschweiften Klammern sind nicht Teil des Wortes.

(7) Befehlsersetzung in eckigen Klammern `[]`

Wenn ein Wort eine eckige Klammer `[` enthält, erfolgt eine Befehlsersetzung. Dazu wird der Tcl-Interpreter erneut aufgerufen, um die Zeichen innerhalb der eckigen Klammern `[]` zu interpretieren. Wenn eckige Klammern in einander geschachtelt werden, müssen genau so viele schließende Klammern gesetzt werden wie geöffnet wurden. Das Ergebnis aus der Interpretation der eckigen Klammern wird als Wort an Stelle der eckigen Klammern eingefügt. Die Befehlsersetzung erfolgt nicht innerhalb geschweifter Klammern `{}` (siehe Regel 6).

(8) Variablenersetzung

Wenn ein Wort ein Dollarzeichen `$` enthält und die direkt nachfolgenden Zeichen der unten stehenden Form entsprechen, erfolgt eine Variablenersetzung. Das Dollarzeichen und die nachfolgenden Zeichen werden durch den Wert der Variable ersetzt. Die Variablenersetzung darf folgende Formen haben:

```
$Variablenname
$Variablenname(Index)
${Variablenname}
```

Die Variablenersetzung erfolgt nicht innerhalb geschweifter Klammern `{}` (siehe Regel 6).

(9) Backslash-Substitution

Wenn innerhalb eines Worten ein Backslash `\` vorkommt, erfolgt eine Backslash-Ersetzung. In allen Fällen (außer in den nachstehend aufgeführten Fällen) wird das Backslash-Zeichen verworfen und das direkt nachfolgende Zeichen als normales Zeichen interpretiert, ohne eine besondere Bedeutung. Dadurch ist es z. B. möglich, Anführungszeichen innerhalb eines Textes zu setzen. Die folgenden Backslash-Folgen haben eine besondere Bedeutung:

```
\a Alarmton
\b Backspace
\f Form feed
\n Neue Zeilenanfang
\r Return
\t Tabulator
\v Vertikaler Tabulator-Stopp
\\ Backslash
```

(10) Kommentar

Wenn an einer Stelle, an der der Tcl-Interpreter das erste Zeichen eines Befehls erwartet, ein `#` Zeichen steht, werden die nachfolgenden Zeichen als Kommentar angesehen und vom Tcl-Interpreter nicht interpretiert. Der Kommentar endet mit dem Zeilenende.

(11) Reihenfolge der Ersetzungen

Jedes Zeichen wird vom Tcl-Interpreter genau einmal interpretiert. Die Ersetzungen erfolgen von links nach rechts und jede Ersetzung wird erst komplett ausgeführt bevor das nächste Wort interpretiert wird. So wird z. B. der Befehl

```
set y [set x 1][incr x][incr x]
```

zum Befehl

```
set y 123
```

(12) Ersetzungen und Wortgrenzen

Ersetzungen ändern nicht die Wortgrenzen des Befehls. Wenn z. B. eine Variable als Wert einen Text mit einem Leerzeichen hat, wird der gesamte Text als ein einziges Wort in den Befehl eingesetzt, nicht als zwei Wörter. Beispiel:

```
puts Guten Tag
```

führt zu einem Fehler, weil der Befehl `puts` als zweiten Befehlsbestandteil entweder einen Channel erwartet und es keinen Channel `Guten` gibt oder (ohne Channel) einen Text ohne Leerzeichen.

```
set Variable "Guten Tag"
puts $Variable
```

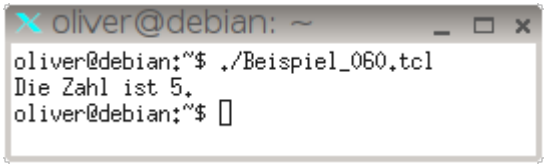
ist stattdessen in Ordnung. Der Tcl-Interpreter macht daraus `puts {Guten Tag}`.

10.3 Ersetzungen

Es ist wichtig, die Ersetzungen bei Anführungszeichen `""`, eckigen Klammern `[]` und geschweiften Klammern `{}` gut zu verstehen. Die meisten tcl-Befehle erwarten ein bestimmte Anzahl an Argumenten. Dabei dient das Leerzeichen als Trennzeichen der einzelnen Argumente. Wenn man aber mehrere Wörter als ein einziges Argument angeben will, dann setzt man die Wörter in Anführungszeichen `""` oder geschweifte Klammern `{}`. Dabei gibt es folgenden Unterschied: Verwendet man Anführungszeichen, dann werden Variablen innerhalb der Anführungszeichen durch den zugeordneten Wert ersetzt. Bei geschweiften Klammern erfolgt keine Ersetzung der Variable. Eckige Klammern `[]` schachteln einen tcl-Befehl in einen anderen tcl-Befehl.

Listing 10.2: Anführungszeichen (Beispiel060.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Zahl 5
4 puts "Die Zahl ist $Zahl."
```



Listing 10.3: Geschweifte Klammern (Beispiel061.tcl)

```
#!/usr/bin/env tclsh

set Zahl 5
puts {Die Zahl ist $Zahl.}
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_061.tcl
Die Zahl ist $Zahl.
oliver@debian:~$
```

Listing 10.4: Anführungszeichen und eckige Klammern (Beispiel062.tcl)

```
#!/usr/bin/env tclsh

set a 1
set b 2
puts "Die Summe aus $a + $b ist [expr $a + $b]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_062.tcl
Die Summe aus 1 + 2 ist 3
oliver@debian:~$
```

Innerhalb der Anführungszeichen werden sowohl die Variablen als auch die in eckigen Klammern gesetzten tcl-Befehle ersetzt.

Listing 10.5: Geschweifte Klammern und eckige Klammern (Beispiel063.tcl)

```
#!/usr/bin/env tclsh

set a 1
set b 2
puts {Die Summe aus $a + $b ist [expr $a + $b]}
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_063.tcl
Die Summe aus $a + $b ist [expr $a + $b]
oliver@debian:~$
```

Bei den geschweiften Klammern findet keine Ersetzung der Variablen und auch nicht der in eckigen Klammern gesetzten tcl-Befehle statt.

11 Listen

11.1 Listen

Eine Liste ist eine Aufzählung von Elementen (Text, Zahlen usw.). Jedes Element steht an einem bestimmten Platz (=Index) in der Liste. Das erste Element hat den Index 0, das zweite Element den Index 1, usw. Das letzte Element hat den Index `end`, das vorletzte Element den Index `end-1`.

Um eine Liste zu initialisieren bzw. um eine Liste zu leeren, verwendet man den Befehl `set Variable {}`.

Mit zunehmender Größe werden Listen (im Unterschied zu Array und Dictionary) langsamer. Bei den heutigen PC-Leistungen kann man dennoch tausende bzw. zehntausende Elemente in einer Liste verwalten, ohne dass die Geschwindigkeit nachlässt.
Bei allen Listenbefehlen (mit Ausnahme `lappend`) wird die ursprüngliche Variable, in der die Liste gespeichert ist, nicht verändert.

Tabelle 11.1: Die wichtigsten Listenbefehle

Befehl	Beschreibung
`set Liste {}`	Initialisiert eine Liste bzw. leert eine bestehende Liste
`set Liste {rot gelb grün}`	Erzeugt eine Liste mit mehreren Elementen.
`set Liste [list rot gelb grün]`	Erzeugt eine Liste mit mehreren Elementen.
`set Liste [list $Farbe1 $Farbe2]`	Erzeugt eine Liste mit mehreren Elementen.
`lappend Liste rot gelb grün`	Fügt der Liste ein oder mehrere Elemente hinzu.
`puts $Liste`	Gibt den Inhalte der Liste aus
`set NeueListe [lreverse $Liste]`	Kehrt die Reihenfolge der Elemente um: das erste Element wird zum letzten Element, das letzte Element wird zum ersten Element.
`set Anzahl [llength $Liste]`	Anzahl der Elemente in der Liste

Tabelle 11.1: Die wichtigsten Listenbefehle

Befehl	Beschreibung

`foreach Element $Liste {`
`  puts $Element`
`}`
	Extrahiert der Reihe nach alle Elemente aus der Liste

`foreach {Element1 Element2} $Liste {`
`  puts "$Element1 Element2"`
`}`
	Extrahiert der Reihe nach immer zwei Elemente gleichzeitig aus der Liste

`puts [lindex $Liste 0]`	Erstes Element
`puts [lindex $Liste 1]`	Zweites Element
`puts [lindex $Liste end-1]`	Vorletztes Element
`puts [lindex $Liste end]`	Letztes Element
`set NeueListe [lrange $Liste 2 5]`	Extrahiert aus der Liste mehrere Elemente. Das Ergebnis ist wieder eine Liste
`lassign $Liste Farbe1 Farbe2`	Speichert die einzelnen Elemente der Liste der Reihen nach in die Variablen Variable1, Variable2 usw. Wenn es mehr Variablen als Elemente gibt, bleiben die restlichen Variablen leer.
`set Rest [lassign $Liste Farbe1 Farbe2]`	Wenn es mehr Elemente als Variablen gibt, gibt der Befehl die restlichen Elemente als Liste zurück. Diese können in einer weiteren Variablen gespeichert werden.

Tabelle 11.1: Die wichtigsten Listenbefehle

Befehl	Beschreibung
`puts [lsearch -all -nocase -inline $Liste rot]`	Sucht in der Liste nach einem Suchbegriff. Wenn die Suche erfolgreich war, wird der Index des ersten Elements zurückgegeben, auf das der Suchbegriff passt. Wenn die Suche nicht erfolgreich war, wird -1 zurückgegeben. Optionen (beliebig kombinierbar): -nocase : Groß-/Kleinschreibung wird nicht beachtet -inline : an Stelle des Index wird das Element zurückgegeben -all : Es wird eine Liste mit allen Indexwerten zurückgegeben, die die Suche erfüllen
`set Liste [lreplace $Liste 0 0]`	Löscht das erste Element
`set Liste [lreplace $Liste 1 3]`	Löscht die Elemente 2 bis 4 (sie haben den Index 1 bis 3)
`set Liste [lreplace $Liste end end]`	Löscht das letzte Element
`set Liste [lreplace $Liste 2 3 schwarz weiß]`	Ersetzt das dritte und vierte Element (Index 2 und 3) durch die Elemente schwarz und weiß.
`set Liste [linsert $Liste 2 schwarz weiß]`	Fügt zwei neue Elemente (schwarz und weiß) an der Index-Position 2 (= drittes Element) ein.
`set Liste [lsort $Liste]`	Sortiert die Liste aufsteigend
`set Liste [lsort -nocase $Liste]`	Sortiert die Liste ohne die Groß-/Kleinschreibung zu beachten
`set Liste [lsort -decreasing $Liste]`	Sortiert die Liste absteigend.
`set Liste [lsort -unique $Liste]`	Sortiert die Liste und entfernt doppelte Einträge
`set Liste [lsort -integer $Liste]`	Sortiert die Liste numerisch (Ganzzahlen)
`set Liste [lsort -real $Liste]`	Sortiert die Liste numerisch (Fließkommazahlen)

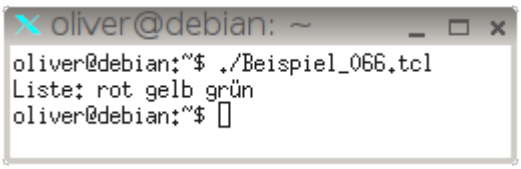
Tabelle 11.1: Die wichtigsten Listenbefehle

Befehl	Beschreibung
`set Liste [lsort -dictionary $Liste]`	Sortiert die Liste alphabetisch
`set Liste [lsort -index 1 $Liste]`	Wenn die Elemente ebenfalls Listen sind, kann man mit der Option -index festlegen, nach welchem Element die Unterlisten sortiert werden soll.
`set Liste [lsort -stride 3 -index 1 $Liste]`	Die Option -stride legt fest, wie viele fortlaufende Elemente der Liste zu einer Gruppe gehören.
`set NeueListe [lmap Element1 $Liste1 Element2 $Liste2 {Befehle}]`	Extrahiert aus den Listen (es können auch mehr als zwei Listen sein) jeweils ein Element, wendet auf die beiden Elemente die Befehle an und gibt das Ergebnis als neue Liste zurück.
`set Liste [concat $Liste1 $Liste2]`	Fügt zwei Listen zu einer Liste zusammen

`package require struct::list`
`struct::list foreachperm Permutation $Liste {`
`  puts $Permutation`
`}`
	Bildet aus den Elementen der Reihe nach alle Kombinationen (Permutation)

Listing 11.1: Eine Liste erzeugen (Beispiel066.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Liste {}
4 set Liste {rot gelb gruen}
5 puts "Liste: $Liste"
```



In Zeile 3 wird die Liste initialisiert. Diese Zeile kann man auch weglassen. In Zeile 4 werden drei Elemente zur Liste hinzugefügt.

Listing 11.2: Elemente mit Leerzeichen (Beispiel386.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Liste {}
4 set Liste {"Donald Duck" "Dagobert Duck"}
5 puts "Liste: $Liste"
6 puts [lindex $Liste 0]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_386.tcl
Liste: "Donald Duck" "Dagobert Duck"
Donald Duck
oliver@debian:~$
```

In Zeile 4 werden zwei Namen (bestehend aus Vor- und Nachname) der Liste hinzugefügt. Da die Namen in Anführungszeichen stehen, wird das Listenelement nicht durch das Leerzeichen getrennt. In Zeile 5 werden die beiden Elemente ausgegeben. Dabei werden automatisch Anführungszeichen um jedes Element gesetzt, um die Elemente (mit ihren Leerzeichen) von einander abzugrenzen. Die Anführungszeichen gehören nicht zu den Elementen, wie man an der Ausgabe in Zeile 6 sieht. In Zeile 6 wird nur das erste Element ausgegeben. Man sieht, dass die Anführungszeichen nicht zum Element gehören.

Listing 11.3: Eine Liste erzeugen, aber mit überraschendem Ergebnis (Beispiel067.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Liste {}
4 set Farbe1 rot
5 set Farbe2 gelb
6 set Farbe3 gruen
7 set Liste {$Farbe1 $Farbe2 $Farbe3}
8 puts "Liste: $Liste"
```

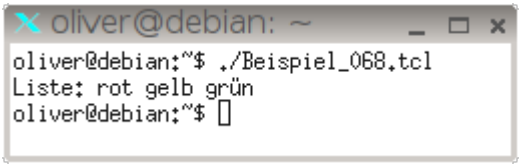
```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_067.tcl
Liste: $Farbe1 $Farbe2 $Farbe3
oliver@debian:~$
```

In den Zeilen 4 bis 6 werden drei Variablen definiert. Deren Inhalt soll in Zeile 4 zu einer Liste zusammengefügt werden. Allerdings werden die Variablen nicht durch ihre Werte ersetzt, weil die Variablen in geschweiften Klammern `{}` stehen und somit der Tcl-Interpreter keine Ersetzung vornimmt.

Listing 11.4: Eine Liste mit dem Befehl list erzeugen (Beispiel068.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Liste {}
```

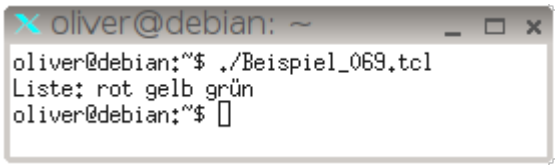
```
4 set Farbe1 rot
5 set Farbe2 gelb
6 set Farbe3 gruen
7 set Liste [list $Farbe1 $Farbe2 $Farbe3]
8 puts "Liste: $Liste"
```



Das Beispiel ähnelt dem vorherigen Beispiel. Durch eckige Klammern `[]` und den Befehl `list` in Zeile 7 werden jetzt aber die Inhalte der Variablen `Farbe1` bis `Farbe3` zu einer Liste zusammengefügt.

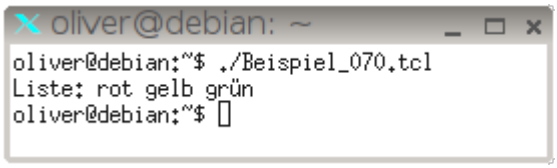
Listing 11.5: Eine Liste mit dem Befehl lappend erzeugen (konstante Werte) (Beispiel069.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Liste {}
4 lappend Liste rot gelb gruen
5 puts "Liste: $Liste"
```



Listing 11.6: Eine Liste mit dem Befehl lappend erzeugen (varibale Werte) (Beispiel070.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Liste {}
4 set Farbe1 rot
5 set Farbe2 gelb
6 set Farbe3 gruen
7 lappend Liste $Farbe1 $Farbe2 $Farbe3
8 puts "Liste: $Liste"
```



Das nächste Beispiel zeigt, dass man aufpassen muss, wenn die Listen-Elemente Leerzeichen enthalten.

Listing 11.7: Vorsicht bei Leerzeichen (Beispiel550.tcl)

```
1  #!/usr/bin/env tclsh
2  set a "Dagobert Duck"
3  set b "Klaas Klever"
4
5  set Liste1 {}
6  lappend Liste1 $a
7  lappend Liste1 $b
8  puts "Liste1:$Liste1"
9  puts "Anzahl Elemente:[llength $Liste1]"
10
11 set Liste2 $a
12 lappend Liste2 $b
13 puts "Liste2:$Liste2"
14 puts "Anzahl Elemente:[llength $Liste2]"
15
16 set Liste3 [list $a]
17 lappend Liste3 $b
18 puts "Liste3:$Liste3"
19 puts "Anzahl Elemente:[llength $Liste3]"
```

In den Zeilen 2 und 3 werden zwei Namen als Listen-Elemente festgelegt. In Zeile 5 wird eine leere `Liste1` erstellt. Die beiden Namen werden anschließend der Liste mit dem Befehl `lappend` hinzugefügt (Zeilen 6 und 7). Im Ergebnis erhält man eine Liste mit zwei Elementen.
In Zeile 11 wird eine zweite Liste erstellt, der sofort bei der Initialisierung der Name aus der Variablen `a` zugeordnet wird. In Zeile 12 wird dann der zweite Name hinzugefügt. Dieses Vorgehen führt zu einer Liste mit drei Elementen, was nicht gewünscht war. Der Grund hierfür war, dass der Inhalt der Variable `a` ein Leerzeichen enthält. Der TCL-Interpreter ersetzt zuerst die Variable `a` durch die beiden Elemente `Dagobert` und `Duck`, so dass der Befehl wie folgt aussieht: `set Liste2 Dagobert Duck`. Somit werden der Liste zwei Elemente hinzugefügt.
Sie können diesen Fehler vermeiden, wenn Sie die Liste immer mit dem Befel `lappend` befüllen (Zeile 6) oder den Befehl `list` verwenden (Zeile 16). In beiden Fällen wird der Inhalt der Variable `a` als ein einziges Element in die Liste eingefügt.

Listing 11.8: Elemente der Liste der Reihe nach aufrufen (Beispiel556.tcl)

```
#!/usr/bin/env tclsh

set Liste {}
set Liste {Anton Dora Berta Caesar}
foreach Element $Liste {
  puts "$Element"
}
```

```
oliver@ubuntu: ~
oliver@ubuntu:~$ ./Beispiel556.tcl
Anton
Dora
Berta
Caesar
oliver@ubuntu:~$
```

Mit dem Befehl `foreach` kann man alle Elemente der Liste der Reihe nach aufrufen.

Listing 11.9: Mehrere Elemente der Liste gleichzeitig der Reihe nach aufrufen (Beispiel557.tcl)

```
#!/usr/bin/env tclsh

set Liste {}
set Liste {Anton 25 Dora 43 Berta 37 Caesar 51}
foreach {Name Alter} $Liste {
  puts "$Name ist $Alter Jahre alt."
}
```

```
oliver@ubuntu: ~
oliver@ubuntu:~$ ./Beispiel557.tcl
Anton ist 25 Jahre alt.
Dora ist 43 Jahre alt.
Berta ist 37 Jahre alt.
Caesar ist 51 Jahre alt.
oliver@ubuntu:~$
```

Man kann auch mehrere Elemente der Liste gleichzeitig aufrufen. Dazu gibt man beim `foreach`-Befehl mehrere Variablen in geschweiften Klammern an.

Listing 11.10: Eine Liste umkehren (Beispiel071.tcl)

```
#!/usr/bin/env tclsh

set Liste {}

lappend Liste Montag
lappend Liste Dienstag
lappend Liste Mittwoch

```

```
9  puts "Liste: $Liste"
10
11 set NeueListe [lreverse $Liste]
12 puts "Neue Liste: $NeueListe"
13
14 puts "Die alte Liste ist weiterhin: $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_071.tcl
Liste: Montag Dienstag Mittwoch
Neue Liste: Mittwoch Dienstag Montag
Die alte Liste ist weiterhin: Montag Dienstag Mittwoch
oliver@debian:~$
```

In Zeile 11 wird mit mit dem Befehl `lreverse` die Reihenfolge der Elemente in der Liste umgekehrt und in einer neuen Variablen gespeichert. Die alte Listenvariable wird dabei nicht verändert (siehe Zeile 14).

Listing 11.11: Listen in Listen (Beispiel073.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {}
4
5  lappend Liste {1 2}
6  lappend Liste {3 4 5}
7  lappend Liste {6 7 8 9}
8
9  puts "Liste: $Liste"
10 puts "Die Liste hat folgende Elemente:"
11 foreach Element $Liste {
12   puts $Element
13 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_073.tcl
Liste: {1 2} {3 4 5} {6 7 8 9}
Die Liste hat folgende Elemente:
1 2
3 4 5
6 7 8 9
oliver@debian:~$
```

In den Zeilen 5 bis 8 werden Zahlenlisten als Elemente der Liste `Liste` hinzugefügt (es werden also Listen in Listen gespeichert). Das erste Element der Liste ist die Liste mit den Zahlen 1 und 2, das zweite Element der Liste ist die Liste mit den Zahlen 3, 4 und 5 und das dritte Element der Liste ist die Liste mit den Zahlen 6, 7, 8 und 9. Hätte man in den Zeilen 5 bis 7 die geschweiften Klammern `{}` weggelassen, wäre jede Zahl als einzelnes Element der Liste hinzugefügt worden und es gäbe keine Listen innerhalb der Liste.

Listing 11.12: Element extrahieren (Beispiel075.tcl)

```
#!/usr/bin/env tclsh

set Liste {}

lappend Liste Montag Dienstag Mittwoch
lappend Liste Donnerstag Freitag
lappend Liste Samstag Sonntag

puts "Liste: $Liste"

puts "Erstes Element:"
puts [lindex $Liste 0]

puts "Zweites Element:"
puts [lindex $Liste 1]

puts "Vorletztes Element:"
puts [lindex $Liste end-1]

puts "Letztes Element:"
puts [lindex $Liste end]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_075.tcl
Liste: Montag Dienstag Mittwoch Donnerstag Freitag Samstag Sonntag
Erstes Element:
Montag
Zweites Element:
Dienstag
Vorletztes Element:
Samstag
Letztes Element:
Sonntag
oliver@debian:~$
```

Listing 11.13: Mehrere fortlaufende Elemente aus der Liste extrahieren (Beispiel076.tcl)

```
#!/usr/bin/env tclsh

set Liste {}

lappend Liste Montag Dienstag Mittwoch Donnerstag
lappend Liste Freitag Samstag Sonntag

puts "Liste: $Liste"
set NeueListe [lrange $Liste 1 3]
puts "Neue Liste: $NeueListe"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_076.tcl
Liste: Montag Dienstag Mittwoch Donnerstag Freitag Samstag Sonntag
Neue Liste: Dienstag Mittwoch Donnerstag
oliver@debian:~$
```

In Zeile 9 werden mit dem Befeh `lrange` drei aufeinander folgende Elemente aus der Liste herausgeholt.

Listing 11.14: Die Elemente einer Liste einzelnen Variablen zuordnen (Beispiel077.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {}
4
5  lappend Liste rot gelb gruen blau
6
7  puts "Liste: $Liste"
8  lassign $Liste Farbe1 Farbe2 Farbe3 Farbe4 Farbe5
9  puts "Farbe1 = $Farbe1"
10 puts "Farbe2 = $Farbe2"
11 puts "Farbe3 = $Farbe3"
12 puts "Farbe4 = $Farbe4"
13 puts "Farbe5 = $Farbe5"
14
15 set RestlicheFarben [lassign $Liste Farbe1 Farbe2]
16 puts "Farbe1 = $Farbe1"
17 puts "Farbe2 = $Farbe2"
18 puts "Restliche Farben = $RestlicheFarben"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_077.tcl
Liste: rot gelb grün blau
Farbe1 = rot
Farbe2 = gelb
Farbe3 = grün
Farbe4 = blau
Farbe5 =
Farbe1 = rot
Farbe2 = gelb
Restliche Farben = grün blau
oliver@debian:~$
```

In Zeile 8 werden die Elemente der Liste den Variablen `Farbe1` bis `Farbe5` zugeordnet. Da die Liste nur vier Elemente hat, bleibt die Variable `Farbe5` leer (siehe Zeile 13). In Zeile 15 werden die Elemente der Liste nur zwei Variablen zugeordnet. Der Rückgabewert des `lassign`-Befehls sind die Listenelemente, die nicht in den Variablen gespeichert wurden. Diese werden in der Variablen `RestlicheFarben` gespeichert.

Listing 11.15: Eine Liste durchsuchen (Beispiel078.tcl)

```
1  #!/usr/bin/env tclsh
```

```
2
3 set Liste {}
4
5 lappend Liste Montag Dienstag Mittwoch Donnerstag
6 lappend Liste Freitag Samstag Sonntag
7
8 puts "Liste: $Liste"
9
10 puts "Mittwoch hat den Index:"
11 puts [lsearch $Liste Mittwoch]
12
13 puts "Samstag hat den Index:"
14 puts [lsearch $Liste Samstag]
15
16 puts "Die Suche nach \"*tag\" findet sechs Elemente (Ausgabe der Elemente als Index):"
17 puts [lsearch -all $Liste *tag]
18
19 puts "Die Suche nach \"*Tag\" findet wegen der Gross-/Kleinschreibung keine Elemente:"
20 puts [lsearch -all $Liste *Tag]
21
22 puts "Die Suche nach \"*Tag\" findet jetzt sechs Elemente (Ausgabe der Elemente als Index):"
23 puts [lsearch -all -nocase $Liste *Tag]
24
25 puts "Die Suche nach \"*Tag\" findet sechs Elemente (Ausgabe der Elemente als Text):"
26 puts [lsearch -all -nocase -inline $Liste *Tag]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_078.tcl
Liste: Montag Dienstag Mittwoch Donnerstag Freitag Samstag Sonntag
Mittwoch hat den Index:
2
Samstag hat den Index:
5
Die Suche nach "*tag" findet sechs Elemente (Ausgabe der Elemente als Index):
0 1 3 4 5 6
Die Suche nach "*Tag" findet wegen der Groß-/Kleinschreibung keine Elemente:

Die Suche nach "*Tag" findet jetzt sechs Elemente (Ausgabe der Elemente als Index):
0 1 3 4 5 6
Die Suche nach "*Tag" findet sechs Elemente (Ausgabe der Elemente als Text):
Montag Dienstag Donnerstag Freitag Samstag Sonntag
oliver@debian:~$
```

Listing 11.16: Elemente löschen (Beispiel079.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Liste {}
4
```

```
5  lappend Liste Montag Dienstag Mittwoch Donnerstag
6  lappend Liste Freitag Samstag Sonntag
7
8  puts "Liste: $Liste"
9
10 set Liste [lreplace $Liste 1 3]
11 puts "Gekuerzte Liste: $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_079.tcl
Liste: Montag Dienstag Mittwoch Donnerstag Freitag Samstag Sonntag
Gekürzte Liste: Montag Freitag Samstag Sonntag
oliver@debian:~$
```

In Zeile 10 werden mit dem Befehl `lreplace` die Elemente mit dem Index 1 bis 3 aus der Liste gelöscht.

Listing 11.17: Elemente ersetzen (Beispiel080.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {}
4
5  lappend Liste rot
6  lappend Liste gelb
7  lappend Liste gruen
8  lappend Liste blau
9
10 puts "Liste: $Liste"
11
12 set Liste [lreplace $Liste 1 2 schwarz weiss]
13 puts "Geaenderte Liste: $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_080.tcl
Liste: rot gelb grün blau
Geänderte Liste: rot schwarz weiß blau
oliver@debian:~$
```

In Zeile 12 werden die Elemente mit dem Index 1 bis 2 durch neue Elemente ersetzt.

Listing 11.18: Elemente einfügen (Beispiel081.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {}
4
5  lappend Liste rot
6  lappend Liste gelb
7  lappend Liste gruen
```

```
8  lappend Liste blau
9
10 puts "Liste: $Liste"
11
12 set Liste [linsert $Liste 2 schwarz weiss]
13 puts "Geaenderte Liste: $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_081.tcl
Liste: rot gelb grün blau
Geänderte Liste: rot gelb schwarz weiß grün blau
oliver@debian:~$
```

In Zeile 12 werden mit dem Befehl `linsert` zwei neue Elemente ab der Index-Position 2 eingefügt.

Listing 11.19: Eine Liste alphabetisch sortieren (Beispiel082.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {}
4
5  lappend Liste Berta Caesar Anton
6
7  puts "Liste: $Liste"
8
9  set Liste [lsort $Liste]
10 puts "Aufsteigend sortiert: $Liste"
11
12 set Liste [lsort -decreasing $Liste]
13 puts "Absteigend sortiert: $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_082.tcl
Liste: Berta Caesar Anton
Aufsteigend sortiert: Anton Berta Caesar
Absteigend sortiert: Caesar Berta Anton
oliver@debian:~$
```

In den Zeilen 9 bzw. 12 wird die Liste sortiert. Durch die Option `-decreasing` in Zeile 12 wird die Liste absteigend sortiert. Die zusätzliche Option `-nocase` ignoriert die Groß-/Kleinschreibung.

Listing 11.20: Eine Liste numerisch sortieren (Beispiel083.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {12 5 18 3 9 11 0 2 1}
4
5  puts "Liste: $Liste"
```

```
6
7  set Liste [lsort $Liste]
8  puts "Aufsteigend alphabetisch sortiert: $Liste"
9
10 set Liste [lsort -integer $Liste]
11 puts "Aufsteigend numerisch sortiert: $Liste"
```

```
oli@debian: ~
oli@debian:~$ ./Beispiel083.tcl
Liste: 12 5 18 3 9 11 0 2 1
Aufsteigend alphabetisch sortiert: 0 1 11 12 18 2 3 5 9
Aufsteigend numerisch sortiert: 0 1 2 3 5 9 11 12 18
oli@debian:~$
```

Durch die Option `-integer` in Zeile 10 wird die Liste numerisch sortiert.

Listing 11.21: Liste mit gruppierten Elementen sortieren (Beispiel425.tcl)

```
1  #!/usr/bin/env tclsh
2
3  # Name Alter Groesse(cm)
4  set Liste {Dora 30 171 Emil 34 188 Berta 28 167 Anton 34 ↩
     190 Caesar 26 187 }
5
6  puts "nach Name sortiert:"
7  puts [lsort -stride 3 -index 0 $Liste]
8
9  puts "nach Alter sortiert:"
10 puts [lsort -stride 3 -index 1 $Liste]
11
12 puts "nach Alter und Name sortiert:"
13 set tmpListe [lsort -stride 3 -index 0 $Liste]
14 puts [lsort -stride 3 -index 1 $tmpListe]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_425.tcl
nach Name sortiert:
Anton 34 190 Berta 28 167 Caesar 26 187 Dora 30 171 Emil 34 188
nach Alter sortiert:
Caesar 26 187 Berta 28 167 Dora 30 171 Emil 34 188 Anton 34 190
nach Alter und Name sortiert:
Caesar 26 187 Berta 28 167 Dora 30 171 Anton 34 190 Emil 34 188
oliver@debian:~$
```

Die Liste besteht aus jeweils drei gruppierten Elementen: `Name`, `Alter`, `Größe` (in cm). Beim Sortieren sollen diese drei Elemente immer zusammengehalten werden. Dazu verwendet man die Option `-stride`. In Zeile 4 wird die Liste erzeugt. In Zeile 7 wird die Liste nach `Name` (erstes Element in der Gruppe) sortiert. Dazu wird mit der Option `-stride 3` festgelegt, dass immer drei Elemente zu einer Gruppe gehören. Mit der Option

`-index 0` wird bestimmt, dass nach dem ersten Element in der Gruppe sortiert wird. In Zeile 10 wird durch die Option `-index 1` nach dem zweiten Element in der Gruppe (`Alter`) sortiert. Wenn man nach mehrere Kriterien gleichzeitig sortieren möchte, muss man den `lsort`-Befehl mehrfach ausführen. Man sortiert zuerst nach dem innersten Sortierkriterium, danach nach dem äußeren. In Zeile 13 wird die Liste zunächst nach `Name` sortiert und anschließend in Zeile 14 nach `Alter`. Im Ergebnis ist die Liste dann nach Alter und innerhalb des Alters nach Namen sortiert.

Listing 11.22: Liste mit Unterlisten sortieren (Beispiel379.tcl)

```
1  #!/usr/bin/env tclsh
2
3  # Liste hat die Elemente: Name Ort Alter
4  set Person1 {Anton Hamburg 28}
5  set Person2 {Dora Berlin 32}
6  set Person3 {Berta Frankfurt 36}
7  set Person4 {Caesar Berlin 29}
8  set Person5 {Emil Berlin 31}
9
10 set Liste {}
11 lappend Liste $Person1
12 lappend Liste $Person2
13 lappend Liste $Person3
14 lappend Liste $Person4
15 lappend Liste $Person5
16
17 puts "unsortiert:"
18 puts $Liste
19 puts ""
20
21 puts "sortiert nach Name:"
22 set ListeSortiert [lsort $Liste]
23 puts $ListeSortiert
24 puts ""
25
26 puts "nach Ort:"
27 set ListeSortiert [lsort -index 1 $Liste]
28 puts $ListeSortiert
29 puts ""
30
31 puts "nach Alter:"
32 set ListeSortiert [lsort -index 2 -integer $Liste]
33 puts $ListeSortiert
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_379.tcl
unsortiert:
{Anton Hamburg 28} {Dora Berlin 32} {Berta Frankfurt 36} {Caesar Berlin 29} {Emil Berlin 31}

sortiert nach Name:
{Anton Hamburg 28} {Berta Frankfurt 36} {Caesar Berlin 29} {Dora Berlin 32} {Emil Berlin 31}

nach Ort:
{Dora Berlin 32} {Caesar Berlin 29} {Emil Berlin 31} {Berta Frankfurt 36} {Anton Hamburg 28}

nach Alter:
{Anton Hamburg 28} {Caesar Berlin 29} {Emil Berlin 31} {Dora Berlin 32} {Berta Frankfurt 36}
oliver@debian:~$
```

Die Liste besteht besteht aus mehreren Personen. Jede Person ist ihrerseits eine Liste mit Name, Ort und Alter. In der Zeile 27 wird die Liste nach dem Ort sortiert. Der Ort hat bei der Personen-Liste den Index 1. In der Zeile 32 wird die Liste nach dem Alter (Index 2) sortiert.

Listing 11.23: Liste mit Unterlisten nach mehreren Kriterien sortieren (Beispiel380.tcl)

```
1  #!/usr/bin/env tclsh
2
3  # Liste hat die Elemente: Name Ort Alter
4  set Person1 {Anton Hamburg 28}
5  set Person2 {Dora Berlin 32}
6  set Person3 {Berta Frankfurt 36}
7  set Person4 {Caesar Berlin 29}
8  set Person5 {Emil Berlin 31}
9
10 set Liste {}
11 lappend Liste $Person1
12 lappend Liste $Person2
13 lappend Liste $Person3
14 lappend Liste $Person4
15 lappend Liste $Person5
16
17 puts "unsortiert:"
18 puts $Liste
19 puts ""
20
21 puts "nach Ort und innerhalb des Ortes nach Name:"
22 set tmpListe [lsort -index 0 $Liste] ; # innere Sortierung
       = nach Name
23 set ListeSortiert [lsort -index 1 $tmpListe] ; # aeussere
      Sortierung = nach Ort
24 puts $ListeSortiert
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_380.tcl
unsortiert:
{Anton Hamburg 28} {Dora Berlin 32} {Berta Frankfurt 36} {Caesar Berlin 29} {Emil Berlin 31}

nach Ort und innerhalb des Ortes nach Name:
{Caesar Berlin 29} {Dora Berlin 32} {Emil Berlin 31} {Berta Frankfurt 36} {Anton Hamburg 28}
oliver@debian:~$
```

In Zeile 22 wird die Liste zunächst nach dem Namen sortiert. Dies ist die innere Sortierung. In Zeile 23 wird dann die neue Liste nach dem Ort sortiert (äußere Sortierung). Da der Sortier-Algorithmus von Tcl die Listenreihenfolge beibehält, kann man nacheinander nach mehreren Kriterien sortieren.

Listing 11.24: Dubletten aus einer Liste entfernen (Beispiel084.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {}
4
5  lappend Liste Berta
6  lappend Liste Caesar
7  lappend Liste Berta
8  lappend Liste Anton
9  lappend Liste Berta
10
11 puts "Die Liste enthaelt Berta dreimal: $Liste"
12
13 set Liste [lsort -unique $Liste]
14 puts "Die Liste enthaelt jedes Element nur noch einmal: ↩
     $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_084.tcl
Die Liste enthält Berta dreimal: Berta Caesar Berta Anton Berta
Die Liste enthält jedes Element nur noch einmal: Anton Berta Caesar
oliver@debian:~$
```

In Zeile 13 wird die Liste sortiert. Durch die Option `-unique` werden dabei die Dubletten entfernt. In Kapitel 11.2 auf Seite 118 sehen Sie, wie man Dubletten entfernen kann, ohne die Reihenfolge der Elemente zu verändern.

Listing 11.25: Zwei Listen zusammenfügen (Beispiel085.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste1 {}
4  set Liste2 {}
5
```

```
6  lappend Liste1 Montag Dienstag Mittwoch Donnerstag
7  lappend Liste2 Freitag Samstag Sonntag
8
9  puts "Liste 1: $Liste1"
10 puts "Liste 2: $Liste2"
11 set Liste [concat $Liste1 $Liste2]
12 puts "Zusammengefuegte Liste: $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_085.tcl
Liste 1: Montag Dienstag Mittwoch Donnerstag
Liste 2: Freitag Samstag Sonntag
Zusammengefügte Liste: Montag Dienstag Mittwoch Donnerstag Freitag Samstag Sonntag
oliver@debian:~$
```

In Zeile 11 werden die beiden Listen `Liste1` und `Liste2` zu einer Liste zusammengefasst.

Listing 11.26: Zwei Listen ohne Dubletten zusammenfügen (Beispiel086.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste1 {}
4  set Liste2 {}
5  set Liste3 {}
6
7  lappend Liste1 Montag Dienstag Mittwoch Donnerstag
8  lappend Liste2 Montag Dienstag Freitag Samstag Sonntag
9  lappend Liste3  Freitag Sonntag
10
11 puts "Liste 1: $Liste1"
12 puts "Liste 2: $Liste2"
13 puts "Liste 3: $Liste3"
14 set Liste [lsort -unique [concat $Liste1 $Liste2 $Liste3]]
15 puts "Zusammengefuegte Liste ohne Dubletten: $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_086.tcl
Liste 1: Montag Dienstag Mittwoch Donnerstag
Liste 2: Montag Dienstag Freitag Samstag Sonntag
Liste 3: Freitag Sonntag
Zusammengefügte Liste ohne Dubletten: Dienstag Donnerstag Freitag Mittwoch Montag Samstag Sonntag
oliver@debian:~$
```

In Zeile 14 wird der `concat`-Befehl, der die drei Listen zusammenfasst, in einen `lsort`-Befehl gesetzt. Der `lsort`-Befehl verwendet die Option `-unique`, um die Dubletten zu löschen. Allerdings ist man vielleicht über die Reihenfolge der Elemente in der zusammengefassten Liste überrascht: Die Tage sind scheinbar durcheinander. Aber man muss daran denken, dass die Liste alphabetisch sortiert wurden.

Man kann Listen nicht nur alphabetisch oder numerisch sortieren, sondern auch gemäß der Vorgabe durch eine zweite Liste. Dazu benutzt man den Befehl `lmap`. Der Befehl extrahiert aus einer oder mehreren Listen jeweils ein Element und wendet darauf einen oder mehrere Befehle an. Die Rückgabe der Befehle kann wieder in eine Liste gespreichert werden. In der einfachsten Form funktioniert der `lmap`-Befehl wie eine `foreach`-Schleife, indem ein Element nach dem anderen aus einer Liste durchlaufen wird.

Listing 11.27: Einfaches Beispiel zum Befehl lmap (Beispiel515.tcl)

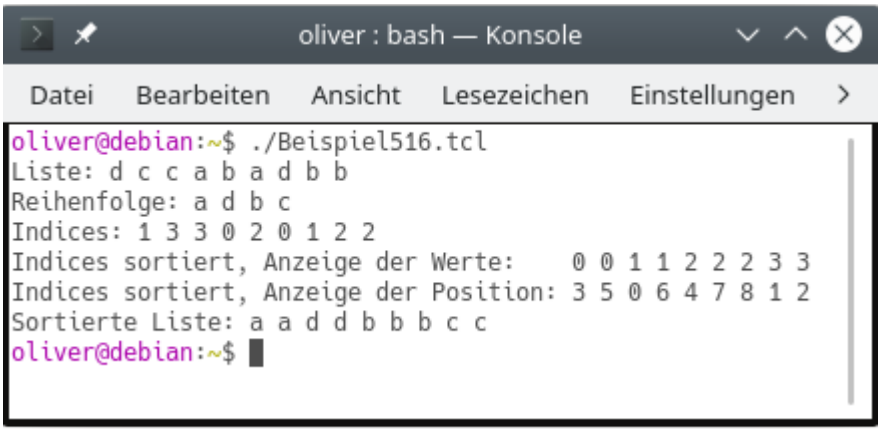
```
1  #!/usr/bin/env tclsh
2
3  set Liste {a b c d e}
4  lmap Element $Liste {puts $Element}
5
6  puts "------------"
7
8  set Liste {a b c d e}
9  set Ergebnis [lmap Element $Liste {list $Element}]
10 puts $Ergebnis
11
12 puts "------------"
13
14 set Liste1 {a b c d e}
15 set Liste2 {1 2 3}
16 set Ergebnis [lmap Element1 $Liste1 Element2 $Liste2 {list
   $Element1 $Element2}]
17 puts $Ergebnis
```

In Zeile 3 wird ein Liste definiert. In Zeile 4 steht der `lmap`-Befehl. Der Befehl durchläuft alle Elemente der Liste, speichert das Element in die Variable `Element` und wendet dann einen (oder mehrere Befehle) auf dass Element an. In diesem Fall wird das Element angezeigt. In Zeile 9 wird der `lmap`-Befehl erweitert, so dass die Rückgabe des Befehls als neue Liste gespeichert wird. Der Befehl hat alle Elemente zu einer neuen Liste zusammengefügt. Bis jetzt unterscheidet sich der Befehl noch nicht von einer `foreach`-Schleife.

In den Zeilen 14 und 15 werden zwei Listen definiert. In Zeile 16 extrahiert der `lmap`-Befehl aus beiden Listen jeweils ein Element und kombiniert beide Elemente zu einem neuen Listen-Element. Alle Kombinationen werden als neue Liste zurückgegeben und in die Variable `Ergebnis` gespeichert. Wie man sieht, müssen die beiden ursprünglichen Listen nicht gleich viel Elemente haben. Wenn in einer Liste weniger Elemente vorkommen als in der anderen Liste, kombiniert der `lmap`-Befehl ein leeres Element.

Listing 11.28: Eine Liste gemäß der Reihenfolge einer zweiten Liste sortieren (Beispiel516.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {d c c a b a d b b}
4  set Reihenfolge {a d b c}
5  puts "Liste: $Liste"
6  puts "Reihenfolge: $Reihenfolge"
7  set Indices [lmap Element $Liste {lsearch -exact ↩
       $Reihenfolge $Element}]
8  puts "Indices: $Indices"
9  set SortiertWerte [lsort -integer $Indices]
10 puts "Indices sortiert, Anzeige der Werte:    ↩
       $SortiertWerte"
11 set SortiertPosition [lsort -indices -integer $Indices]
12 puts "Indices sortiert, Anzeige der Position: ↩
       $SortiertPosition"
13 set SortierteListe [lmap Position $SortiertPosition {↩
       lindex $Liste $Position}]
14 puts "Sortierte Liste: $SortierteListe"
```



In Zeile 3 wird eine Liste definiert, die sortiert werden soll. Die Reihenfolge, wie die Elemente sortiert werden sollen, gibt eine zweite Liste vor (Zeile 4).
In Zeile 7 erstellt der `lmap`-Befehl eine Liste, in der jedes Element der ursprünglichen Liste die Indexposition aus der Reihenfolge-Liste bekommt. Das bedeutet: das erste Element der Liste ist der Buchstabe `d`. In der Reihenfolge-Liste hat der Buchstabe `d` die Indexposition `1`. Der nächste Buchstabe `c` hat die Indexposition `3`. Somit wird korrespondieren zur Liste `{d c c a b a d b b}` die Indexpositionen `{1 3 3 0 2 0 1 2 2}`. Wenn man diese Indices-Liste jetzt normal sortieren würde (wie in Zeile 9), bekäme man die Liste

{0 0 1 1 2 2 2 3 3}. Stattdessen sortiert man die Indices-Liste mit der zusätzlichen Option -indices (Zeile 11). Das bewirkt, dass das rangniedrigste Element (also der Wert 0) in der Indices-Liste an der Position 3 vorkommt. Das im Rang nächstfolgende Element (wieder der Wert 0) kommt in der Indices-Liste an Position 5 vor. Das im Rang darauf folgende Element [es hat den Wert 1] steht in der Indices-Liste an Position 0. Somit ergibt sich folgende Liste: {3 5 0 6 4 7 8 1 2}. In Zeile 13 wird mit dieser Liste die ursprüngliche Liste sortiert und die sortierte Liste in Zeile 14 angezeigt.

11.2 Dubletten entfernen ohne die Reihenfolge zu ändern

Mit dem Befehl lsort -unique kann man die Dubletten aus einer Liste entfernen. Der Nachteil ist, dass die Liste dabei immer sortiert wird. Nachfolgend sehen Sie eine Prozedur, die die Dubletten entfernt, aber die Reihenfolge der Elemente beibehält.

Listing 11.29: Dubletten entfernen ohne die Reihenfolge zu ändern (Beispiel552.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc ListeDuplikateEntfernen {Liste} {
4          # Entfernt aus der Liste die Duplikate, ohne die ↩
             Reihenfolge der Listenelemente zu aendern
5          for {set i 0} {$i < [llength $Liste]} {incr i} {
6                  set Indizes [lsearch -all $Liste [lindex ↩
                     $Liste $i]]
7                  for {set j [expr [llength $Indizes] - 1]} ↩
                     {$j > 0} {incr j -1} {
8                          # die doppelten Listenelemente vom↩
                               Ende aus entfernen
9                          set Index [lindex $Indizes $j]
10                         set Liste [lreplace $Liste $Index ↩
                               $Index]
11                 }
12         }
13         return $Liste
14 }
15
16 set Liste {Caesar Dora Anton Berta Dora Anton Emil Caesar}
17 puts "Liste: $Liste"
18 set ListeOhneDuplikate [ListeDuplikateEntfernen $Liste]
19 puts "Liste ohne Duplikate unter Beibehaltung der ↩
     Reihenfolge:"
20 puts $ListeOhneDuplikate
```

```
oliver@ubuntu: ~
oliver@ubuntu:~$ ./Beispiel552.tcl
Liste: Caesar Dora Anton Berta Dora Anton Emil Caesar
Liste ohne Duplikate unter Beibehaltung der Reihenfolge:
Caesar Dora Anton Berta Emil
oliver@ubuntu:~$
```

11.3 Unterschied zwischen lappend und concat

Beide Befehle `lappend` und `concat` fügen Elemente zu einer Liste hinzu. Aber die Befehle führen zu unterschiedlichen Ergebnissen.

Listing 11.30: lappend und concat (Beispiel087.tcl)

```
#!/usr/bin/env tclsh

set Liste {Anton Berta Caesar}
lappend Liste {Dora Emil Friedrich}
lappend Liste {Gustav Heinrich Ida}
puts "Mit dem Befehl lappend sieht es so aus:"
puts $Liste
puts "4. Element = [lindex $Liste 3]"
puts "5. Element = [lindex $Liste 4]"

set Liste {Anton Berta Caesar}
set Liste [concat $Liste {Dora Emil Friedrich}]
set Liste [concat $Liste {Gustav Heinrich Ida}]
puts "Mit dem Befehl concat sieht es so aus:"
puts $Liste
puts "4. Element = [lindex $Liste 3]"
puts "5. Element = [lindex $Liste 4]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_087.tcl
Mit dem Befehl lappend sieht es so aus:
Anton Berta Cäsar {Dora Emil Friedrich} {Gustav Heinrich Ida}
4. Element = Dora Emil Friedrich
5. Element = Gustav Heinrich Ida
Mit dem Befehl concat sieht es so aus:
Anton Berta Cäsar Dora Emil Friedrich Gustav Heinrich Ida
4. Element = Dora
5. Element = Emil
oliver@debian:~$
```

Der Befehl `lappend` fügt die Listen als Ganzes zusammen. Somit hat die Liste fünf Elemente. Das vierte Element ist eine Liste aus den drei Namen `Dora`, `Emil` und `Friedrich`. Und auch das fünfte Element ist eine Liste mit drei Namen. Der Befehl `concat` fügt jedes Element der Listen einzeln zur Gesamtliste zusammen. Somit hat die Liste zum Schluss neun Elemente. Ergänzender Hinweis: `concat` löst keine eingebetteten Listen, also Listen in Listen, auf.

Listing 11.31: lappend und concat (Beispiel088.tcl)

```
#!/usr/bin/env tclsh

set Liste1 {}
set Liste2 {}

set Liste1 {Montag Dienstag Mittwoch}
set Liste2 {Donnerstag Freitag}
```

```
8
9  puts "Liste 1: $Liste1"
10 puts "Liste 2: $Liste2"
11
12 lappend Liste4 $Liste1
13 lappend Liste4 $Liste2
14 puts "Liste, mit lappend zusammengefuegt: $Liste4"
15 puts "Anzahl Elemente in der Liste: [llength $Liste4]"
16 puts "Index 0: [lindex $Liste4 0]"
17 puts "Index 1: [lindex $Liste4 1]"
18 puts "Index 2: [lindex $Liste4 2]"
19 puts "Index 3: [lindex $Liste4 3]"
20 puts "Index 4: [lindex $Liste4 4]"
21
22 set Liste3 [concat $Liste1 $Liste2]
23 puts "Liste, mit concat zusammengefuegt: $Liste3"
24 puts "Anzahl Elemente in der Liste: [llength $Liste3]"
25 puts "Index 0: [lindex $Liste3 0]"
26 puts "Index 1: [lindex $Liste3 1]"
27 puts "Index 2: [lindex $Liste3 2]"
28 puts "Index 3: [lindex $Liste3 3]"
29 puts "Index 4: [lindex $Liste3 4]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_088.tcl
Liste 1: Montag Dienstag Mittwoch
Liste 2: Donnerstag Freitag
Liste, mit lappend zusammengefügt: {Montag Dienstag Mittwoch} {Donnerstag Freitag}
Anzahl Elemente in der Liste: 2
Index 0: Montag Dienstag Mittwoch
Index 1: Donnerstag Freitag
Index 2:
Index 3:
Index 4:
Liste, mit concat zusammengefügt: Montag Dienstag Mittwoch Donnerstag Freitag
Anzahl Elemente in der Liste: 5
Index 0: Montag
Index 1: Dienstag
Index 2: Mittwoch
Index 3: Donnerstag
Index 4: Freitag
oliver@debian:~$
```

11.4 Elemente aus einer Liste extrahieren mit foreach

Befehl:

```
foreach Element $Liste {
 Anweisung
}
```

Der Befehl `foreach` durchläuft die gesamte Liste und holt der Reihe nach jedes einzelne Element aus der Liste heraus.

Listing 11.32: Elemente einzeln aus der Liste herausholen (Beispiel072.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {}
4
5  lappend Liste Montag Dienstag Mittwoch
6
7  puts "Die Liste hat folgende Elemente:"
8  foreach Element $Liste {
9    puts $Element
10 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_072.tcl
Die Liste hat folgende Elemente:
Montag
Dienstag
Mittwoch
oliver@debian:~$
```

In Zeile 8 wird mit dem Befehl `foreach` jedes Element einzeln aus der Liste herausgeholt und in Zeile 9 angezeigt.

Listing 11.33: Elemente paarweise aus der Liste herausholen (Beispiel074.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {}
4
5  lappend Liste 1 1 2 4 3 9 4 16
6
7  puts "Liste: $Liste"
8  foreach {x y} $Liste {
9    puts "Das Quadrat von $x ist $y."
10 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_074.tcl
Liste: 1 1 2 4 3 9 4 16
Das Quadrat von 1 ist 1.
Das Quadrat von 2 ist 4.
Das Quadrat von 3 ist 9.
Das Quadrat von 4 ist 16.
oliver@debian:~$
```

In Zeile 8 werden die Elemente der Liste paarweise aus der Liste herausgeholt. Dazu schreibt man das Variablenpaar in geschweifte Klammern.

Listing 11.34: Mehrere Elemente gleichzeitig aus mehreren Listen herausholen (Beispiel319.tcl)

```
1  #!/usr/bin/env tclsh
```

```
2
3 set Namen {Anton Berta Caesar Dora}
4 set Orte {Augsburg Bonn Chemnitz Dresden}
5
6 foreach Name $Namen Ort $Orte {
7   puts "$Name / $Ort"
8 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_319.tcl
Anton / Augsburg
Berta / Bonn
Cäsar / Chemnitz
Dora / Dresden
oliver@debian:~$
```

Man kann auch gleichzeitig aus mehreren Listen die Elemente herausholen. In Zeile 6 werden die Elemente der Reihe nach gleichzeitig sowohl aus der Liste `Namen` als auch aus der Liste `Orte` extrahiert. Es wird aus beiden Listen zunächst das erste Element geholt, danach aus beiden Listen das zweite Element usw.

11.5 Vergleich von Listen mit ::struct::set

Befehle:

- `package require struct::set`
- `[::struct::set union $Liste1 $Liste2 $Liste3]`
- `[::struct::set intersect $Liste1 $Liste2 $Liste3]`
- `[::struct::set difference $Liste1 $Liste2]`
- `[::struct::set symdiff $Liste1 $Liste2]`
- `[::struct::set equal $Liste1 $Liste2]`

Das Paket `struct::set` stellt weitere Listen-Befehle bereit, mit denen man die Vereinigungs-, Schnitt- oder Differenzmenge aus zwei oder mehr Listen erstellen kann. Außerdem kann man prüfen, ob zwei Listen identisch sind.

Listing 11.35: Vergleich von Listen mit ::struct::set (Beispiel509.tcl)

```
1 #!/usr/bin/env tclsh
2 package require struct::set
3
4 set Liste1 {1 3 4 7 8}
5 set Liste2 {1 2 4 5 6 7}
6 set Liste3 {4 7 9}
7 set Liste4 {1 3 4 7 8}
8
```

```
9  puts "Liste1: $Liste1"
10 puts "Liste2: $Liste2"
11 puts "Liste3: $Liste3"
12 puts "Liste4: $Liste4"
13 puts ""
14 puts "Vereinigungsmenge Liste1 bis Liste3:"
15 puts [lsort [::struct::set union $Liste1 $Liste2 $Liste3]]
16 puts ""
17 puts "Schnittmenge Liste1 bis Liste3:"
18 puts [lsort [::struct::set intersect $Liste1 $Liste2 ↩
   $Liste3]]
19 puts ""
20 puts "Elemente von Liste1, die nicht in Liste2 vorkommen:"
21 puts [lsort [::struct::set difference $Liste1 $Liste2]]
22 puts ""
23 puts "Elemente von Liste1, die nicht in Liste2 vorkommen ↩
   sowie"
24 puts "die Elemente von Liste2, die nicht in Liste1 ↩
   vorkommen:"
25 puts [lsort [::struct::set symdiff $Liste1 $Liste2]]
26 puts ""
27 puts "Prueft, ob Liste1 identisch ist mit Liste2:"
28 puts [::struct::set equal $Liste1 $Liste2]
29 puts "Prueft, ob Liste1 identisch ist mit Liste4:"
30 puts [::struct::set equal $Liste1 $Liste4]
```

```
oli@debian: ~
oli@debian:~$ ./Beispiel509.tcl
Liste1: 1 3 4 7 8
Liste2: 1 2 4 5 6 7
Liste3: 4 7 9
Liste4: 1 3 4 7 8

Vereinigungsmenge Liste1 bis Liste3:
1 2 3 4 5 6 7 8 9

Schnittmenge Liste1 bis Liste3:
4 7

Elemente von Liste1, die nicht in Liste2 vorkommen:
3 8

Elemente von Liste1, die nicht in Liste2 vorkommen sowie
die Elemente von Liste2, die nicht in Liste1 vorkommen:
2 3 5 6 8

Prueft, ob Liste1 identisch ist mit Liste2:
0
Prueft, ob Liste1 identisch ist mit Liste4:
1
oli@debian:~$
```

11.6 Alle Kombinationen aus den Elementen einer Liste (Permutation)

Befehle:

```
package require struct::list
struct::list foreachperm Permutation $Liste {
 Anweisungen
}
```

Eine Permutation ist die Kombination von Elementen in einer bestimmten Reihenfolge. So kann man die Elemente A B C wie folgt kombinieren: ABC, ACB, BAC, BCA, CAB und CBA. Wenn es n Elemente gibt, gibt es n! (Fakultät) Permutationen. Dies bedeutet bei vier Elementen gibt es 4*3*2*1 = 24 Kombinationen. Der Befehl `foreachperm` erstellt der Reihe nach alle Kombinationen aus den Elementen einer Liste.

Listing 11.36: Permutation (Beispiel382.tcl)

```
1 #!/usr/bin/env tclsh
2
3 package require struct::list
4
5 set Liste {1 2 3}
6 struct::list foreachperm Permutation $Liste {
7   puts $Permutation
8 }
```

In Zeile 3 wird das Paket `struct::list` eingebunden. Das Paket ist Teil von `tcllib`. In Zeile 5 wird eine Liste mit drei Elementen erzeugt. In Zeile 6 werden der Reihe nach alle Kombinationen aus den Elementen gebildet und in Zeile 7 angezeigt.

11.7 Listen in Listen (eingebettete Listen)

Man kann auch ganze Listen in Listen einfügen.

Listing 11.37: Listen in Listen (Beispiel089.tcl)

```
#!/usr/bin/env tclsh

set Gruppe1 {Anton Berta Caesar}
set Gruppe2 {Dora Emil}
set Gruppe3 {Friedrich Gustav Heinrich Ida}
```

```
6
7 set Gruppen [list $Gruppe1 $Gruppe2 $Gruppe3]
8 puts "Alle Gruppen:"
9 puts $Gruppen
10
11 puts "Die Mitglieder der Gruppe 2:"
12 puts [lindex $Gruppen 1]
13
14 puts "Das zweite Mitglied der Gruppe 3:"
15 puts [lindex [lindex $Gruppen 2] 1]
16 puts [lindex $Gruppen 2 1]
```

```
oli@debian: ~
oli@debian:~$ ./Beispiel089.tcl
Alle Gruppen:
{Anton Berta Caesar} {Dora Emil} {Friedrich Gustav Heinrich Ida}
Die Mitglieder der Gruppe 2:
Dora Emil
Das zweite Mitglied der Gruppe 3:
Gustav
Gustav
oli@debian:~$
```

In den Zeilen 3 bis 5 werden drei Listen definiert. In Zeile 7 werden mit dem `list`-Befehl drei Listen zu einer Gruppenliste zusammengefasst. Das erste Element der Gruppenliste ist dann die `Gruppe1`, das zweite Element die `Gruppe2` usw. In Zeile 12 wird das zweite Element (=Index 1) aus der Gruppenliste angezeigt. In Zeile 15 wird in der innersten Klammer das dritte Element (=Index 2) aus der Gruppenliste gewählt. Aus dieser Liste wird dann in der äußeren Klammer das zweite Element (=Index 1) angezeigt. In Zeile 16 ist ein vereinfachter Zugriff dargestellt.

11.8 Listen expandieren mit {*}

Befehl:

- `{*}$Variable`

In manchen Fällen werden die Leerzeichen zwischen den einzelnen Elementen einer Liste nicht als Trennzeichen betrachtet. In diesen Fällen kann man die Liste expandieren, so dass jedes Element der Liste zu einem einzelnen Argument wird. Dazu schreibt man direkt vor die Listenvariable ein `{*}`.

Listing 11.38: Dateien kopieren (so funktioniert es nicht) (Beispiel376.tcl)

```
1 #!/usr/bin/env tclsh
2
3 file mkdir tmp
4 set Liste [glob *.txt]
5 file copy $Liste tmp
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_376.tcl
error copying "Datei1.txt Datei2.txt": no such file or directory
    while executing
"file copy $Liste tmp"
    (file "./Beispiel_376.tcl" line 5)
oliver@debian:~$
```

In Zeile 3 wird das Verzeichnis `tmp` erstellt. In Zeile 4 werden alle Textdateien im aktuellen Ordner gesucht und die Dateinamen als Liste gespeichert. Dies sind im Beispiel die Dateien `Datei1.txt` und `Datei2.txt`. In Zeile 5 sollen diese Dateien in das Verzeichnis `tmp` kopiert werden. Doch das funktioniert nicht und es erscheint eine Fehlermeldung. Beim Versuch die Dateien zu kopieren, wurden die beiden Dateinamen nicht als einzelne Dateien erkannt, sondern es sollte eine Datei mit dem Namen `"Datei1.txt Datei2.txt"` kopiert werden. Eine solche Datei gibt es aber nicht.

Listing 11.39: Dateien kopieren (jetzt funktioniert es) (Beispiel377.tcl)

```
1 #!/usr/bin/env tclsh
2
3 file mkdir tmp
4 set Liste [glob *.txt]
5 file copy {*}$Liste tmp
```

In Zeile 5 wurde `{*}` direkt vor die Listenvariable `$Liste` geschrieben. Dadurch wird zuerst die Liste expandiert, so dass die einzelnen Elemente der Liste zu einzelnen Elementen im `copy`-Befehl werden. Der Befehl, der dadurch ausgeführt wird, lautet jetzt `file copy Datei1.txt Datei2.txt tmp`.

12 Text

12.1 Text

Bei Tcl/Tk ist im Unterschied zu anderen Programmiersprachen alles ein Text. Selbst Zahlen werden intern sowohl als Zahl als auch als Text gespeichert. Text wird bei den meisten Programmiersprachen String genannt. Das erste Zeichen im Text hat den Index `0`, das zweite Zeichen den Index `1`. Das letzte Zeichen hat den Index `end`, das vorletzte Zeichen den Index `end-1`.

Tabelle 12.1: Die wichtigsten Textbefehle

Befehl	Beschreibung
`puts [string length $Text]`	Anzahl der Zeichen
`puts [string cat $Text1 "abc"]`	Fügt zwei oder mehr Textvariablen oder Texte zusammen
`puts [string index $Text 0]`	Das erste Zeichen
`puts [string index $Text 1]`	Das zweite Zeichen
`puts [string index $Text end-1]`	Das vorletzte Zeichen
`puts [string index $Text end]`	Das letzte Zeichen
`puts [string range $Text 1 3]`	Das zweite bis vierte Zeichen
`puts [string range $Text 1 end]`	Der Text ohne das erste Zeichen
`puts [string range $Text 0 end-1]`	Der Text ohne das letzte Zeichen
`puts [string first a $Text]`	Position, an der ein Zeichen oder ein Suchbegriff das erste Mal vorkommt. Gibt den Wert -1 zurück, wenn der Suchbegriff nicht gefunden wird. Die Groß-/ Kleinschreibung wird beachtet.
`puts [string last a $Text]`	Position, an der ein Zeichen oder ein Suchbegriff das letzte Mal vorkommt. Gibt den Wert -1 zurück, wenn der Suchbegriff nicht gefunden wird. Die Groß-/ Kleinschreibung wird beachtet.

Tabelle 12.1: Die wichtigsten Textbefehle

Befehl	Beschreibung
`puts [string match "*kurz*" $Text]`	Sucht den Suchbegriff in dem Text und gibt den Wert 1 zurück, wenn der Suchbegriff im Text enthalten ist. Wenn der Suchbegriff nicht im Text vorkommt, wird der Wert 0 zurückgegeben. Die Groß-/ Kleinschreibung wird beachtet.
`puts [string match -nocase \` `"*Kurz*" $Text]`	Die Option `-nocase` legt fest, dass die Groß-/ Kleinschreibung nicht beachtet wird.
`puts [string match "k?rz" $Text]`	Das `?` ersetzt genau ein beliebiges Zeichen
`puts [string match "k*z" $Text]`	Das `*` ersetzt beliebig viele Zeichen
`set Text [string replace $Text 4 9` `"langer"]`	Ersetzt einen Teil des Textes durch einen neuen Text. Im Beispiel werden die Buchstaben 5 bis 10 durch das Wort `langer` ersetzt.
`puts [string map {a u r l} Hammer]`	Ersetzt in dem Text die Buchstaben bzw. Zeichen durch andere Buchstaben bzw. Zeichen gemäß der Schlüssel-Wert-Paare. Aus `Hammer` wird `Hummel`. Die Groß-/ Kleinschreibung wird beachtet. Auf diese Art dürfen die Schlüssel-Wert-Paare keine Variablen enthalten. Für die Verwendung von Variablen, siehe die nächsten Befehle.

`puts [string map [list $Buchstabe1`
`$Buchstabe2 $Buchstabe3`
`$Buchstabe4] Hammer]`
	Durch den `list`-Befehl dürfen die Schlüssel-Wert-Paare Variablen enthalten.

`set Liste {}`
`lappend Liste $Buchstabe1`
`$Buchstabe2 $Buchstabe3`
`$Buchstabe4`
`puts [string map $Liste Hammer]`
	Eine weitere Möglichkeit, die Schlüssel-Wert-Paare mit Variablen zu bilden, ist, zuerst eine Liste anzulegen.

`puts [string map -nocase {A u r l}` `Hammer]`	Die Option `-nocase` ignoriert die Groß-/ Kleinschreibung.
`puts [string repeat Text Anzahl]`	Wiederholt den Text

Tabelle 12.1: Die wichtigsten Textbefehle

Befehl	Beschreibung
`puts [string tolower $Text]`	Ersetzt alle Großbuchstaben durch Kleinbuchstaben
`puts [string toupper $Text]`	Ersetzt alle Kleinbuchstaben durch Großbuchstaben
`puts [string trim $Text]`	Löscht alle Leerzeichen am Anfang und am Ende des Textes
`puts [string trim $Text z]`	Löscht ein bestimmtes Zeichen am Anfang und am Ende des Textes. Im Beispiel werden alle z am Anfang und am Ende gelöscht.
`puts [string trimleft $Text]`	Löscht alle Leerzeichen am Anfang des Textes
`puts [string trimright $Text]`	Löscht alle Leerzeichen am Ende des Textes

Listing 12.1: Länge eines Strings (Beispiel090.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "Ein kurzer Text"
4 set Ganzzahl 12345
5 set Kommazahl 123.45
6
7 puts "$Text hat [string length $Text] Zeichen."
8 puts "$Ganzzahl hat [string length $Ganzzahl] Ziffern."
9 puts "$Kommazahl hat [string length $Kommazahl] Stellen ↩
    inklusive Dezimalpunkt."
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_090.tcl
Ein kurzer Text hat 15 Zeichen.
12345 hat 5 Ziffern.
123.45 hat 6 Stellen inklusive Dezimalpunkt.
oliver@debian:~$
```

Listing 12.2: Zeichen wiederholen (Beispiel522.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "Text"
4
```

```
5 puts [string repeat "a" 5]
6 puts [string repeat $Text 5]
```

Listing 12.3: Groß-/Kleinbuchstaben (Beispiel523.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "Das ist ein Text."
4
5 puts [string tolower $Text]
6 puts [string toupper $Text]
```

Listing 12.4: (Leer-)Zeichen am Anfang/Ende entfernen (Beispiel524.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "  Leerzeichen    "
4 puts "===$Text==="
5 puts "===[string trimleft $Text]==="
6 puts "===[string trimright $Text]==="
7 puts "===[string trim $Text]==="
8
9 puts ""
10
11 set Text "xxZeichenxxxx"
12 set Zeichen "x"
13 puts "===$Text==="
14 puts "===[string trimleft $Text $Zeichen]==="
15 puts "===[string trimright $Text $Zeichen]==="
16 puts "===[string trim $Text $Zeichen]==="
```

```
oliver@debian:~$ ./Beispiel524.tcl
===  Leerzeichen    ===
===Leerzeichen    ===
===  Leerzeichen===
===Leerzeichen===

===xxZeichenxxxx===
===Zeichenxxxx===
===xxZeichen===
===Zeichen===
oliver@debian:~$
```

Listing 12.5: Texte bzw. Textvariablen verknüpfen (Beispiel415.tcl)

```tcl
#!/usr/bin/env tclsh

set a "abc"
set b "def"
set c "ghi"
puts [string cat $a $b $c "jkl"]
```

```
oliver@debian:~$ ./Beispiel_415.tcl
abcdefghijkl
oliver@debian:~$
```

Listing 12.6: Ein Zeichen extrahieren (Beispiel091.tcl)

```tcl
#!/usr/bin/env tclsh

set Text "Ein kurzer Text"
puts $Text
puts "Erstes Zeichen: [string index $Text 0]"
puts "Zweites Zeichen: [string index $Text 1]"
puts "Vorletztes Zeichen: [string index $Text end-1]"
puts "Letztes Zeichen: [string index $Text end]"
```

```
oliver@debian:~$ ./Beispiel_091.tcl
Ein kurzer Text
Erstes Zeichen: E
Zweites Zeichen: i
Vorletztes Zeichen: x
Letztes Zeichen: t
oliver@debian:~$
```

Listing 12.7: Mehrere Zeichen extrahieren (Beispiel092.tcl)

```
#!/usr/bin/env tclsh

set Text "Ein kurzer Text"
puts $Text
puts "Das sechste bis achte Zeichen: [string range $Text 5 7]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_092.tcl
Ein kurzer Text
Das sechste bis achte Zeichen: urz
oliver@debian:~$
```

Listing 12.8: Ohne das erste Zeichen (Beispiel298.tcl)

```
#!/usr/bin/env tclsh

set Text "Ein kurzer Text"
puts $Text
puts "Der Text ohne das erste Zeichen: [string range $Text 1 end]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_298.tcl
Ein kurzer Text
Der Text ohne das erste Zeichen: in kurzer Text
oliver@debian:~$
```

Listing 12.9: Ohne das letzte Zeichen (Beispiel095.tcl)

```
#!/usr/bin/env tclsh

set Text "Ein kurzer Text"
puts $Text
puts "Der Text ohne das letzte Zeichen: [string range $Text 0 end-1]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_095.tcl
Ein kurzer Text
Der Text ohne das letzte Zeichen: Ein kurzer Tex
oliver@debian:~$
```

Listing 12.10: Ein Zeichen finden (erstmaliges Vorkommen) (Beispiel096.tcl)

```
#!/usr/bin/env tclsh
```

```
2
3 set Text "Ein kurzer Text"
4 puts $Text
5 puts "Das erste e hat folgenden Index: [string first e ↩
    $Text]"
6 puts "Der Suchbegriff urz kommt an folgendem Index zum ↩
    ersten Mal vor: [string first urz $Text]"
```

```
oliver@debian: ~
Ein kurzer Text
Das erste e hat folgenden Index: 8
Der Suchbegriff urz kommt an folgendem Index zum ersten Mal vor: 5
oliver@debian:~$
```

Da die Groß-/ Kleinschreibung beachtet wird, kommt das erste `e` nicht schon als Index `0` vor, sondern erst als Index `8`.

Listing 12.11: Ein Zeichen finden (alle Positionen) (Beispiel097.tcl)

```
1 #!/usr/bin/env tclsh
2
3 proc StringFindAll {Text Suchbegriff} {
4   set Liste {}
5   set Index [string last $Suchbegriff $Text]
6   while {$Index >= 0} {
7     lappend Liste $Index
8     set Text [string range $Text 0 [expr $Index -1]]
9     set Index [string last $Suchbegriff $Text]
10  }
11  set Liste [lsort -integer $Liste]
12  return $Liste
13 }
14
15 set Text "Ein kurzer Text"
16 puts $Text
17
18 set Suchbegriff "e"
19 set Liste [StringFindAll $Text $Suchbegriff]
20 puts "\"$Suchbegriff\" kommt an folgenden Positionen (↩
    Index) vor: $Liste"
21
22 set Suchbegriff "kur"
23 set Liste [StringFindAll $Text $Suchbegriff]
24 puts "\"$Suchbegriff\" kommt an folgenden Positionen (↩
    Index) vor: $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_097.tcl
Ein kurzer Text
"e" kommt an folgenden Positionen (Index) vor: 8 12
"kur" kommt an folgenden Positionen (Index) vor: 4
oliver@debian:~$
```

In Zeile 4 wird eine leere Liste erzeugt, die alle Index-Positionen aufnehmen soll, an denen der Suchbegriff gefunden wird. In Zeile 5 wird die letzte Position im Text gesucht, an der der Suchbegriff vorkommt. Wenn der Suchbegriff vorkommt, wird die `while`-Schleife aufgerufen (Zeile 6). In Zeile 7 wird die Index-Position der Liste hinzugefügt. In Zeile 8 wird der Text vom Ende her verkürzt. In Zeile 9 wird die letzte Position im (verbliebenen) Text gesucht, an der der Suchbegriff vorkommt. In Zeile 11 wird die Liste numerisch aufsteigend sortiert. In Zeile 12 wird die Liste an das Hauptprogramm zurückgegeben.

Listing 12.12: Ein Zeichen finden (alle Positionen, mit Listen-Befehlen) (Beispiel098.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Text "Ein kurzer Text"
4  puts $Text
5
6  set TextAlsListe [split $Text ""]
7  set Liste [lsearch -all $TextAlsListe e]
8  puts "Das e kommt an folgenden Positionen (Index) vor: ↩
     $Liste"
9
10 set TextAlsListe [split $Text " "]
11 set Liste [lsearch -all $TextAlsListe kurzer]
12 puts "Das Wort \"kurzer\" kommt an folgenden Positionen ↩
     Index) vor: $Liste"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_098.tcl
Ein kurzer Text
Das e kommt an folgenden Positionen (Index) vor: 8 12
Das Wort "kurzer" kommt an folgenden Positionen (Index) vor: 1
oliver@debian:~$
```

In Zeile 6 wandelt der `split`-Befehl den Text in eine Liste um. Jeder Buchstabe ist ein Element der Liste. In Zeile 7 wird die Liste mit dem Befehl `lsearch` und der Option `-all` nach allen `e` durchsucht. In Zeile 10 wandelt der `split`-Befehl den Text erneut in eine Liste um. Da als Trennzeichen ein Leerzeichen verwendet wird, wird jedes Wort zu einem Element der Liste. In Zeile 11 wird die Liste mit dem Befehl `lsearch` und der Option `-all` nach allen Wörtern `kurzer` durchsucht. In Zeile 12 werden die Index-Positionen ausgegeben. Es handelt sich jetzt aber um den Index bezüglich der Wörter, nicht der Buchstaben.

Listing 12.13: Prüfen, ob der Text einen bestimmten Suchbegriff enthält (Beispiel099.tcl)

```
#!/usr/bin/env tclsh

puts "Im Folgenden bedeutet 0=nicht gefunden und 1=gefunden."

set Text "Ein kurzer Text"
puts $Text

puts "Der Text enthaelt den Suchbegriff \"kurz\": [string match "*kurz*" $Text]"
puts "Der Text enthaelt nicht den Suchbegriff \"karz\": [string match "*karz*" $Text]"
puts "Der Text enthaelt nicht den Suchbegriff \"Kurz\" (wegen der Gross-/Kleinschreibung): [string match "*Kurz*" $Text]"
puts "Der Text enthaelt den Suchbegriff \"Kurz\", weil die Gross/Kleinschreibung jetzt ignoriert wird: [string match -nocase "*Kurz*" $Text]"
puts "Der Text enthaelt den Suchbegriff \"k?rz\": [string match "*k?rz*" $Text]"
puts "Der Text enthaelt nicht den Suchbegriff \"k?z\", weil das Fragezeichen genau ein Zeichen ersetzt: [string match "*k?z*" $Text]"
puts "Der Text enthaelt den Suchbegriff \"k??z\", weil jedes Fragezeichen genau ein Zeichen ersetzt: [string match "*k??z*" $Text]"
puts "Der Text enthaelt den Suchbegriff \"k*z\", weil das Sternchen beliebig viele Zeichen ersetzt: [string match "*k*z*" $Text]"
```

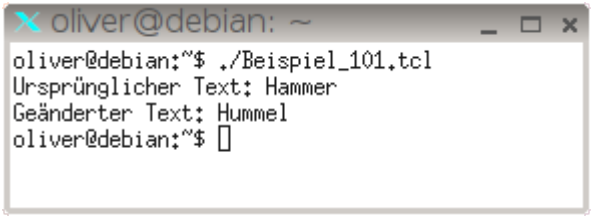
```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_099.tcl
Im Folgenden bedeutet 0=nicht gefunden und 1=gefunden.
Ein kurzer Text
Der Text enthält den Suchbegriff "kurz": 1
Der Text enthält nicht den Suchbegriff "karz": 0
Der Text enthält nicht den Suchbegriff "Kurz" (wegen der Groß-/Kleinschreibung): 0
Der Text enthält den Suchbegriff "Kurz", weil die Groß/Kleinschreibung jetzt ignoriert wird: 1
Der Text enthält den Suchbegriff "k?rz": 1
Der Text enthält nicht den Suchbegriff "k?z", weil das Fragezeichen genau ein Zeichen ersetzt: 0
Der Text enthält den Suchbegriff "k??z", weil jedes Fragezeichen genau ein Zeichen ersetzt: 1
Der Text enthält den Suchbegriff "k*z", weil das Sternchen beliebig viele Zeichen ersetzt: 1
oliver@debian:~$
```

Wenn der Suchbegriff gefunden wird, ist der Rückgabewert `1`. Sonst ist der Rückgabewert `0`.

Listing 12.14: Zeichen ersetzen mit dem Befehl string map (ohne Variablen) (Beispiel101.tcl)

```
#!/usr/bin/env tclsh

set Text "Hammer"
```

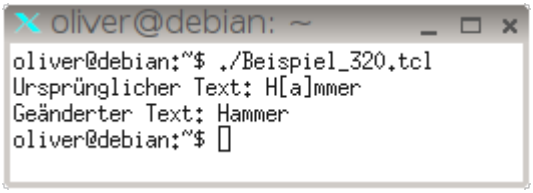
```
4 puts "Urspruenglicher Text: $Text"
5
6 set Text [string map {a u r l} $Text]
7 puts "Geaenderter Text: $Text"
```



Der Befehl `string map` ersetzt Buchstaben im Text. Die Schlüssel-Wert-Paare sind: `a` wird zu `u` und `r` wird zu `l`. Somit wird aus `Hammer` `Hummel`.

Listing 12.15: Eckige Klammern (und andere Sonderzeichen) ersetzen (Beispiel320.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "H\[a\]mmer"
4 puts "Urspruenglicher Text: $Text"
5
6 set Text [string map {"\[" "" "\]" ""} $Text]
7 puts "Geaenderter Text: $Text"
```



In Zeile 3 werden eckige Klammern `[` und `]` in einen Text eingefügt. Die eckigen Klammern müssen mit einem Schrägstrich \ maskiert werden, damit sie nicht als Befehl interpretiert werden. In Zeile 6 werden die eckigen Klammern `[` und `]` durch `""` ersetzt. Hierbei müssen die eckigen Klammern ebenfalls mit einem Schrägstrich \ maskiert werden. Andere Sonderzeichen wie geschweifte Klammern `{}`, Anführungszeichen `""` usw. können ebenso ersetzt werden, wenn man sie maskiert.
Die Schlüssel-Wert-Paare beim `string map`-Befehl stellen eine Liste dar. Wenn man statt konstanter Buchstaben/Zeichen mit Variablen arbeiten möchte, darf man die Schlüssel-Wert-Paare nicht in geschweifte Klammern schreiben. Stattdessen muss man den `list`-Befehl verwenden oder vor dem `string map`-Befehl eine Liste erstellen und die Listen-Variable verwenden. Beides wird im nächsten Beispiel gezeigt.

Listing 12.16: Zeichen ersetzen mit dem Befehl string map und Variablen (Beispiel561.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "Hammer"
4 puts "Urspruenglicher Text: $Text"
```

```
5  puts ""
6
7  set Buchstabe1 "a"
8  set Buchstabe2 "i"
9  set Buchstabe3 "r"
10 set Buchstabe4 "l"
11
12 puts "Mit geschweiften Klammern geht es nicht:"
13 set Text [string map {$Buchstabe1 $Buchstabe2 $Buchstabe3 ↩
       $Buchstabe4} $Text]
14 puts "Unveraenderter Text: $Text"
15 puts ""
16
17 puts "Mit dem list-Befehl geht es:"
18 set Text [string map [list $Buchstabe1 $Buchstabe2 ↩
       $Buchstabe3 $Buchstabe4] $Text]
19 puts "Geaenderter Text: $Text"
20 puts ""
21
22 puts "Mit einer Liste geht es:"
23 set Liste {}
24 lappend Liste $Buchstabe1 $Buchstabe2 $Buchstabe3 ↩
       $Buchstabe4
25 set Text [string map $Liste $Text]
26 puts "Geaenderter Text: $Text"
```

```
oliver@ubuntu: ~
oliver@ubuntu:~$ ./Beispiel561.tcl
Urspruenglicher Text: Hammer

Mit geschweiften Klammern geht es nicht:
Unveraenderter Text: Hammer

Mit dem list-Befehl geht es:
Geaenderter Text: Himmel

Mit einer Liste geht es:
Geaenderter Text: Himmel
oliver@ubuntu:~$
```

In den Zeilen 7 bis 10 werden die Ersetzungen in Variablen gespeichert. In Zeile 13 werden die Variablen in geschweiften Klammern als Schlüssel-Wert-Paare angegeben. Da der Tcl-Interpreter Variablen in geschweiften Klammern nicht ersetzt (siehe Kapitel 10 auf Seite 91), funktioniert der `string map`-Befehl nicht. In Zeile 18 wird statt der geschweiften Klammern der `list`-Befehl verwendet. Jetzt funktioniert der `string map`-Befehl. Alternativ kann man auch erst eine Liste erstellen (Zeilen 23 und 24) und dann statt der geschweiften Klammern die Listen-Variable einsetzen (Zeile 25).

12.2 Text verketten

Befehl 1:
```
set VariableNeu $Variable1$Variable2
set VariableNeu ${Variable1}${Variable2}
```

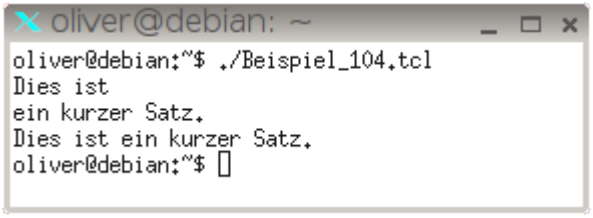
Befehl 2:
```
set VariableNeu ""
append VariableNeu $Variable1
append VariableNeu $Variable2
```

Befehl 3 (erst ab Tcl 8.6):
```
set VariableNeu [string cat $Variable1 $Variable2]
```

Mit dem Befehl `string cat` kann man mehrere Texte bwz. Text-Variablen zu einem Text zusammenfassen. Der Befehl wurde erst mit der Tcl-Version 8.6 eingeführt. Daneben gibt es noch zwei weitere Möglichkeiten, um zum gleichen Ergebnis zu gelangen: man schreibt die Variablen direkt hintereinander oder verwendet den Befehl `append`. Diese beiden Varianten funktionieren auch mit früheren Tcl-Versionen.

Listing 12.17: Variablen direkt hintereinander (Beispiel104.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text1 "Dies ist "
4 set Text2 "ein kurzer Satz."
5 puts $Text1
6 puts $Text2
7 set NeuerText $Text1$Text2
8 puts $NeuerText
```



Listing 12.18: Verketten mit append (Beispiel105.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text1 "Dies ist "
4 set Text2 "ein kurzer Satz."
5 puts $Text1
6 puts $Text2
7 set NeuerText ""
8 append NeuerText $Text1
9 append NeuerText $Text2
10 puts $NeuerText
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_105.tcl
Dies ist
ein kurzer Satz.
Dies ist ein kurzer Satz.
oliver@debian:~$
```

Der Befehl `append` hängt einen Text direkt an einen bestehenden Text an (Zeilen 8 und 9). Die Zeile 7 kann auch entfallen, weil der Befehl `append` die Variable neu anlegt, wenn sie noch nicht existiert. Der Programmcode ist aber verständlicher, wenn die Variable separat definiert wird.

12.3 Text zerlegen

Befehl:

- `split $Variable Trennzeichen`

Der `split`-Befehl zerlegt einen Text anhand eines oder mehrerer Trennzeichen in einzelne Elemente (z. B. Wörter, Buchstaben, Zahlen, Ziffern). Wird kein Trennzeichen angegeben, wird als Trennzeichen ein `whitespace` (das sind Leerzeichen, Tabulator und NewLine) verwendet. Die Elemente können dann als Liste gespeichert werden.

Listing 12.19: Mehrzeiligen Text in einzelne Zeilen zerlegen (Beispiel106.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "Die erste Zeile.\nDie zweite Zeile.\nDie dritte ↩
    Zeile."
4 puts $Text
5 set Liste [split $Text "\n"]
6 puts "Zeile 1: [lindex $Liste 0]"
7 puts "Zeile 2: [lindex $Liste 1]"
8 puts "Zeile 3: [lindex $Liste 2]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_106.tcl
Die erste Zeile.
Die zweite Zeile.
Die dritte Zeile.
Zeile 1: Die erste Zeile.
Zeile 2: Die zweite Zeile.
Zeile 3: Die dritte Zeile.
oliver@debian:~$
```

In Zeile 4 wird ein mehrzeiliger Text erzeugt. In der Regel wird dies eine Textdatei sein. In Zeile 5 wird der Text in die einzelnen Zeilen getrennt und in der Variablen Liste gespeichert. Das Trennzeichen `\n` trennt den Text am Zeilenende.

Listing 12.20: Mehrzeiligen Text in einzelne Wörter zerlegen (Beispiel107.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "Die erste Zeile.\nDie zweite Zeile.\nDie dritte ↩
    Zeile."
4 puts $Text
5 set Liste [split $Text]
6 puts "Wort 1: [lindex $Liste 0]"
7 puts "Wort 2: [lindex $Liste 1]"
8 puts "Wort 3: [lindex $Liste 2]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_107.tcl
Die erste Zeile.
Die zweite Zeile.
Die dritte Zeile.
Wort 1: Die
Wort 2: erste
Wort 3: Zeile.
oliver@debian:~$
```

In Zeile 5 wurde kein Trennzeichen angegeben. Deshalb wird als Trennzeichen ein sogenanntes whitespace verwendet: Dies sind Leerzeichen, Tabulator und NewLine.

Listing 12.21: Einzeiligen Text in einzelne Wörter zerlegen (Beispiel108.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "Anton Berta Caesar Dora"
4 puts "Text: $Text"
5 set Liste [split $Text]
6 puts "Liste: $Liste"
7 puts "Element 1: [lindex $Liste 0]"
8 puts "Element 2: [lindex $Liste 1]"
9 puts "Element 3: [lindex $Liste 2]"
10 puts "Element 4: [lindex $Liste 3]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_108.tcl
Text: Anton Berta Cäsar Dora
Liste: Anton Berta Cäsar Dora
Element 1: Anton
Element 2: Berta
Element 3: Cäsar
Element 4: Dora
oliver@debian:~$
```

Listing 12.22: Text in einzelne Wörter zerlegen (ein Trennzeichen) (Beispiel109.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Text "Anton,Berta,Caesar,Dora"
```

```
4  puts "Text: $Text"
5  set Liste [split $Text ","]
6  puts "Liste: $Liste"
7  puts "Element 1: [lindex $Liste 0]"
8  puts "Element 2: [lindex $Liste 1]"
9  puts "Element 3: [lindex $Liste 2]"
10 puts "Element 4: [lindex $Liste 3]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_109.tcl
Text: Anton,Berta,Cäsar,Dora
Liste: Anton Berta Cäsar Dora
Element 1: Anton
Element 2: Berta
Element 3: Cäsar
Element 4: Dora
oliver@debian:~$
```

In Zeile 5 wird der Text in einzelne Elemente zerlegt. Als Trennzeichen wird das Komma verwendet.

Listing 12.23: Text in einzelne Wörter zerlegen (mehrere Trennzeichen) (Beispiel110.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Text "Anton:Berta.Caesar,Dora"
4  puts "Text: $Text"
5  set Liste [split $Text ",.:"]
6  puts "Liste: $Liste"
7  puts "Element 1: [lindex $Liste 0]"
8  puts "Element 2: [lindex $Liste 1]"
9  puts "Element 3: [lindex $Liste 2]"
10 puts "Element 4: [lindex $Liste 3]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_110.tcl
Text: Anton:Berta.Cäsar,Dora
Liste: Anton Berta Cäsar Dora
Element 1: Anton
Element 2: Berta
Element 3: Cäsar
Element 4: Dora
oliver@debian:~$
```

In Zeile 5 wird der Text in einzelne Elemente zerlegt. Dabei werden mehrere Trennzeichen berücksichtigt. Die Trennzeichen sind in Anführungszeichen `""` oder in geschweifte Klammern `{}` zu setzen.

Listing 12.24: Text in einzelne Buchstaben zerlegen (Beispiel111.tcl)

```
1  #!/usr/bin/env tclsh
2
```

```
3  set Text "Anton"
4  puts "Text: $Text"
5  set Liste [split $Text ""]
6  puts "Liste: $Liste"
7  puts "Element 1: [lindex $Liste 0]"
8  puts "Element 2: [lindex $Liste 1]"
9  puts "Element 3: [lindex $Liste 2]"
10 puts "Element 4: [lindex $Liste 3]"
11 puts "Element 5: [lindex $Liste 4]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_111.tcl
Text: Anton
Liste: A n t o n
Element 1: A
Element 2: n
Element 3: t
Element 4: o
Element 5: n
oliver@debian:~$
```

In Zeile 5 wird der Text in einzelne Buchstaben zerlegt. Als Trennzeichen werden leere Anführungszeichen `""` oder leere geschweifte Klammern `{}` verwendet.

12.4 Text zusammenfügen

Befehl:

- `set Variable [join $Liste Trennzeichen]`

Der `join`-Befehl ist die Umkehrung des `split`-Befehls. Der `join`-Befehl fasst die einzelnen Elemente einer Liste zu einem Gesamttext zusammen. Dabei wird ein Trennzeichen zwischen die einzelnen Listenelemente eingefügt.

Listing 12.25: Elemente einer Liste zu einem Text zusammenfügen (Beispiel113.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Liste {Anton Berta Caesar Dora}
4  puts "Liste: $Liste"
5  set Text [join $Liste ,]
6  puts "Text: $Text"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_113.tcl
Liste: Anton Berta Cäsar Dora
Text: Anton,Berta,Cäsar,Dora
oliver@debian:~$
```

In Zeile 5 werden die drei Elemente der Liste zu einem Gesamttext zusammengefasst. Die Elemente werden mit einem Komma getrennt.

12.5 Text überprüfen

Befehl:

- `string is Klasse $Zeichen`

Der Befehl `string is` überpüft, ob das Zeichen einer bestimmten Klasse (z. B. Zahlen 0-9, Buchstaben a-z) angehört. Der Befehl gibt den Wert `1` zurück, wenn das Zeichen der Klasse angehört. Sonst ist der Wert `0`.

Tabelle 12.2: Die wichtigsten Klassen

Klasse	Beschreibung
`alnum`	Buchstabe oder Zahl
`alpha`	Buchstabe
`integer`	Zahl (Ganzzahl)
`double`	Zahl (Gleitkommazahl)
`lower`	Kleinbuchstabe
`upper`	Großbuchstabe

Listing 12.26: Eingabe auf Klasse prüfen (Beispiel112.tcl)

```
1  #!/usr/bin/env tclsh
2
3  puts "Geben Sie ein Zeichen oder einen Text ein:"
4  set Zeichen [gets stdin]
5
6  if {[string is alpha $Zeichen]} {
7    puts "$Zeichen besteht nur aus Buchstaben."
8  }
9
10 if {[string is integer $Zeichen]} {
11   puts "$Zeichen besteht nur aus Zahlen."
12 }
13
14 if {[string is lower $Zeichen]} {
15   puts "$Zeichen besteht nur aus Kleinbuchstaben."
16 }
17
18 if {[string is upper $Zeichen]} {
19   puts "$Zeichen besteht nur aus Grossbuchstaben."
20 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_112.tcl
Geben Sie ein Zeichen oder einen Text ein:
aBc
aBc besteht nur aus Buchstaben.
oliver@debian:~$
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_112.tcl
Geben Sie ein Zeichen oder einen Text ein:
ABC
ABC besteht nur aus Buchstaben.
ABC besteht nur aus Großbuchstaben.
oliver@debian:~$
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_112.tcl
Geben Sie ein Zeichen oder einen Text ein:
abc
abc besteht nur aus Buchstaben.
abc besteht nur aus Kleinbuchstaben.
oliver@debian:~$
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_112.tcl
Geben Sie ein Zeichen oder einen Text ein:
123
123 besteht nur aus Zahlen.
oliver@debian:~$
```

Beim Vergleich mit `string is double` muss man beachten, dass nur Zahlen in der Form `-123456.78` als gültige Gleitkommazahl betrachtet werden (als Dezimaltrennzeichen muss der Punkt verwendet werden und es gibt kein Tausendertrennzeichen). Zahlen in landestypischer Schreibweise wie zum Beispiel `-123.456,78` werden stattdessen nicht als gültige Gleitkommazahl angesehen. Hierfür muss man sich eine eigene Prüfprozedur programmieren.

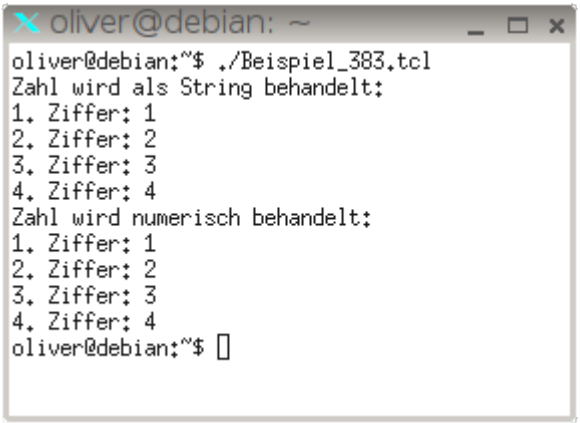
12.6 Zahl als Text oder als Zahl behandeln

In Tcl/Tk kann man Zahlen sowohl als Text als auch als Zahl behandeln. Normalerweise ist es sinnvoll, Zahlen nur numerisch zu verwenden, aber man ist nicht dazu gezwungen.

Listing 12.27: Zahl oder Text (Beispiel383.tcl)

```
1 #!/usr/bin/env tclsh
```

```
2
3 set Zahl1 1000
4 set Zahl2 234
5 set Zahl3 [expr $Zahl1 + $Zahl2]
6
7 puts "Zahl wird als String behandelt:"
8 puts "1. Ziffer: [string index $Zahl3 0]"
9 puts "2. Ziffer: [string index $Zahl3 1]"
10 puts "3. Ziffer: [string index $Zahl3 2]"
11 puts "4. Ziffer: [string index $Zahl3 3]"
12
13 puts "Zahl wird numerisch behandelt:"
14 set Ziffer1 [expr $Zahl3 / 1000]
15 set Ziffer2 [expr ($Zahl3 % 1000) / 100]
16 set Ziffer3 [expr ($Zahl3 % 100) / 10]
17 set Ziffer4 [expr ($Zahl3 % 10)]
18 puts "1. Ziffer: $Ziffer1"
19 puts "2. Ziffer: $Ziffer2"
20 puts "3. Ziffer: $Ziffer3"
21 puts "4. Ziffer: $Ziffer4"
```



In den Zeilen 3 bis 5 wird die Zahl 1234 berechnet und in der Variablen `Zahl3` gespeichert. Die Berechnung macht deutlich, dass es sich tatsächlich um eine Zahl handelt. In den Zeilen 7 bis 11 wird die Variable `Zahl` wie ein Text behandelt, und es wird jede Ziffer wie ein Zeichen aus einem Text extrahiert. In den Zeilen 13 bis 21 wird die Variable `Zahl` wie eine Zahl behandelt, und die Ziffern werden rechnerisch extrahiert. Man sieht, dass es manchmal vorteilhaft ist, eine Zahl wie Text zu behandeln.

12.7 Mustererkennung und Text ersetzen

Befehle:

- `regexp {Muster} Text`
- `regsub -all Suchbegriff Text Variable`

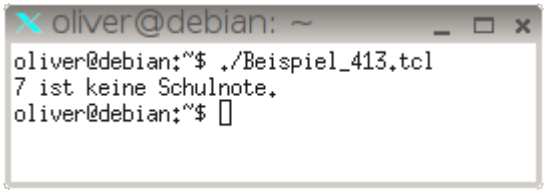
Mit dem bisher vorgestellt Befehl `string match` kann man nur eine einfache Textsuche durchführen. Die beiden Befehle `regexp` und `regsub` sind deutlich leistungsfähiger, weil sie die Syntax regulärer Ausdrücke (regular expression syntax) verwenden. Über diese Syntax gibt es ganze Bücher, so dass im Folgenden nur ein einfaches Beispiel vorgestellt wird. Der Befehl `regexp` führt eine Mustererkennung in Text durch und gibt als Ergebnis eine `0` (das Muster wurde nicht erkannt) oder `1` (das Muster wurde erkannt) zurück. Das Muster, auf das der Text überprüft wird, wird in geschweifte Klammern `{}` gesetzt. Der Befehl hat folgenden Aufbau: `regexp {Muster} Text`. Der Befehl `regsub` ersetzt Textteile und speichert das Ergebnis in einer Variablen ab. Der Befehl hat folgenden Aufbau: `regsub -all Suchbegriff Text Variable`. Die Option `-all` wird verwendet, wenn alle Suchbegriffe im Text ersetzt werden sollen. Wenn die Option weggelassen wird, erfolgt nur beim erstmaligen Auftauchen des Suchbegriffs eine Ersetzung. Nachdem die Ersetzungen im Text vorgenommen wurden, wird das Ergebnis in der Variable gespeichert. Die beiden Befehle ermöglichen auch die Suche nach bzw. das Ersetzen von sogenannten Escape-Sequenzen.

Tabelle 12.3: Die wichtigsten Escape-Sequenzen

Escape-Sequenz	Beschreibung
`\n`	Neue Zeile
`\r`	Return
`\t`	Tabulator
`\"`	Anführungszeichen maskieren

Listing 12.28: Mustererkennung mit regexp (Beispiel413.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Note 7
4
5  set Ergebnis [regexp {[1-6]} $Note]
6  if {$Ergebnis == 1} {
7    puts "$Note ist eine Schulnote."
8   } else {
9    puts "$Note ist keine Schulnote."
10 }
```

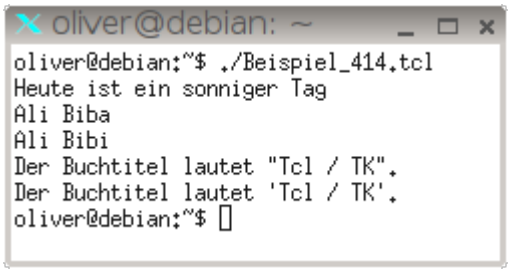


In Zeile 5 wird geprüft, ob es sich um eine gültige Schulnote handelt. Wenn die Prüfung mit `regexp` erfolgreich war, wird der Wert `1` zurückgegeben. Beachten Sie die geschweiften

Klammern `{}` als Teil des `regexp`-Befehls.

Listing 12.29: Text ersetzen mit regsub (Beispiel414.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Text "Heute ist ein regnerischer Tag"
4  regsub "regnerischer" $Text "sonniger" Text
5  puts $Text
6
7  set Text "Ali Baba"
8  regsub "a" $Text "i" Text1
9  puts $Text1
10 regsub -all "a" $Text "i" Text2
11 puts $Text2
12
13 set Text "Der Buchtitel lautet \"Tcl / TK\"."
14 puts $Text
15 regsub -all \" $Text ' Text
16 puts $Text
```



In Zeile 4 wird das Wort `regnerischer` durch `sonniger` ersetzt. Das Ergebnis wird in der Variable `Text` gespeichert. In Zeile 8 wird das erste `a` durch ein `i` ersetzt und der Text in der Variablen `Text1` gespeichert. In Zeile 10 wird durch die Option `-all` bestimmt, dass alle Buchstaben `a` durch ein `i` ersetzt werden sollen. Das Ergebnis wird in der Variablen `Text2` gespeichert. In Zeile 13 wird ein Text mit doppelten Anführungszeichen gespeichert. Beachten Sie das Maskieren der Anführungszeichen mit einem vorangestellten Schrägstrich `\`. In Zeile 15 werden die doppelten Anführungszeichen durch einfache Anführungszeichen ersetzt.

12.8 Platzhalter im Text

Befehl:

- `format["Ein Text mit Platzhalter %s im Satz." $Variable]`

Man kann in einen Text Platzhalter für Zahlen, Zeichen oder anderen Text einfügen. Der Text wird in den Befehl `format[]` geschrieben. Die Platzhalter werden in den Text gesetzt. Der Platzhalter wird während der Ausgabe durch den Wert der Variable ersetzt. Die Variablen werden in der Reihenfolge der Platzhalter an das Ende des Textes geschrieben.

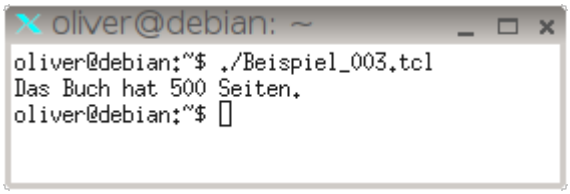
Die Platzhalter sind nützlich, wenn man z. B. Fließkommazahlen in einem bestimmten Format darstellen möchte. Außerdem sind Platzhalter in der Regel notwendig, wenn man Texte mehrsprachig ausgeben möchte.

Tabelle 12.4: Platzhalter

Platzhalter	Beschreibung
`%d`	Ganzzahl
`%.xf`	Fließkommazahl mit `x` Nachkommastellen
`%s`	Text oder Zeichen
`%c`	Zeichen. Es wird der ASCII-Code erwartet (z. B. `65` für den Buchstaben `A`)

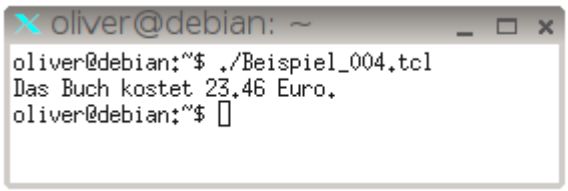
Listing 12.30: Text mit Platzhalter für eine Ganzzahl (Beispiel003.tcl)

```
1 #!/usr/bin/env tclsh
2 set Zahl 500
3 puts [format "Das Buch hat %d Seiten." $Zahl]
```



Listing 12.31: Text mit Platzhalter für eine Fließkommazahl (Beispiel004.tcl)

```
1 #!/usr/bin/env tclsh
2 set Preis 23.45678
3 puts [format "Das Buch kostet %.2f Euro." $Preis]
```



Listing 12.32: Text mit Platzhalter für ein Zeichen (Beispiel005.tcl)

```
1 #!/usr/bin/env tclsh
2 set Zeichen A
3 puts [format "Das Alphabet beginnt mit dem Buchstaben %s."
      $Zeichen]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_005.tcl
Das Alphabet beginnt mit dem Buchstaben A.
oliver@debian:~$
```

Listing 12.33: Text mit Platzhalter für ein Zeichen (ASCII-Code) (Beispiel006.tcl)

```
#!/usr/bin/env tclsh
set Zeichen 65
puts [format "Das Alphabet beginnt mit dem Buchstaben %c." $Zeichen]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_006.tcl
Das Alphabet beginnt mit dem Buchstaben A.
oliver@debian:~$
```

Listing 12.34: Text mit Platzhalter für einen Text (Beispiel007.tcl)

```
#!/usr/bin/env tclsh
set Text "Herr Meier"
puts [format "Sehr geehrter %s," $Text]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_007.tcl
Sehr geehrter Herr Meier,
oliver@debian:~$
```

Listing 12.35: Mehrere Platzhalter im Text (Beispiel008.tcl)

```
#!/usr/bin/env tclsh
set Text "ein Text"
set Ganzzahl 123
set Dezimalzahl 45.6789
puts [format "Dies ist %s. Eine Zahl sieht so %d oder so %.2f aus." $Text $Ganzzahl $Dezimalzahl]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_008.tcl
Dies ist ein Text. Eine Zahl sieht so 123 oder so 45.68 aus.
oliver@debian:~$
```

13 Array

13.1 Array

Ein Array ist eine (Sammel-)Variable, die mehrere Werte unter eindeutigen Element-Namen speichert. Ein Array ist wie ein Schrank mit vielen Schubladen. Jede Schublade hat eine eindeutige Bezeichnung. Über den Namen des Arrays (=Schrank) und den Namen des Elements (=Schublade) kann man auf den Inhalt zugreifen.

Der Name des Elements wird in runde Klammern `()` direkt hinter den Arraynamen geschrieben. Der Name des Elements kann auch aus dem Inhalt einer oder mehrerer Variablen (gegebenenfalls kombiniert mit weiteren Zeichen) gebildet werden, z. B. `Array($Variable)` oder `Array($Variable1,$Variable2)`.

Im Unterschied zu Listen haben die Elemente eines Arrays keine bestimmte Index-Position und werden mit zunehmender Größe auch nicht langsamer.

Mit dem Befehl `parray Variable` kann man das gesamte Array anzeigen. Arrays können nicht wie andere Variablen an Prozeduren übergeben werden. Hierzu muss man den Befehl `upvar` verwenden oder das Array in der Prozedur als `global` deklarieren. Mit dem Befehl `array set Variable {}` kann man ein Array initialisieren. Das ist aber nicht notwendig.

Tabelle 13.1: Die wichtigsten Array-Befehle

Befehl	Beschreibung
`array set Person {}`	Initialisiert ein leeres Array
`set Person(Name) Donald`	Legt den Wert für ein Element im Array fest.
`puts $Person(Name)`	Gibt den Wert aus.

`array set Person {`
`  Vorname Donald`
`  Nachname Duck`
`}`
	Erzeugt ein Array und definiert zugleich die Elemente des Arrays.

`set Liste {Vorname Donald`
`  Nachname Duck}`
`array set Person $Liste`
	Wenn eine Listenvariable aus den Paarungen Element und Wert besteht, kann sie ebenfalls dazu verwendet werden, ein Array zu definieren.

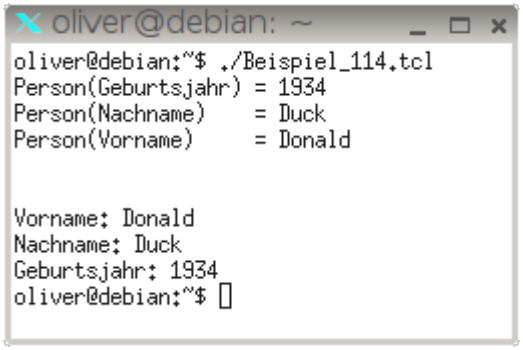
`parray Person`	Zeigt das gesamte Array an

Tabelle 13.1: Die wichtigsten Array-Befehle

Befehl	Beschreibung
`puts [array get Person]`	Gibt das gesamte Array als Liste zurück
`puts [array names Person]`	Gibt die Namen der Elemente zurück
`puts [array size Person]`	Gibt die Anzahl der Elemente zurück

Listing 13.1: Array anlegen (Variante 1) (Beispiel114.tcl)

```
1  #!/usr/bin/env tclsh
2
3  array set Person {}
4  set Person(Vorname) Donald
5  set Person(Nachname) Duck
6  set Person(Geburtsjahr) 1934
7
8  parray Person
9  puts "\n"
10 puts "Vorname: $Person(Vorname)"
11 puts "Nachname: $Person(Nachname)"
12 puts "Geburtsjahr: $Person(Geburtsjahr)"
```



In Zeile 3 wird das Array `Person` initialisiert. Diese Zeile kann man auch weglassen. In Zeile 8 wird das gesamte Array ausgegeben. Zeile 9 erzeugt eine Leerzeile. In den Zeilen 10 bis 12 wird auf einzelne Elemente des Arrays zugegriffen.

Listing 13.2: Array anlegen (Variante 2) (Beispiel372.tcl)

```
#!/usr/bin/env tclsh

array set Person {
  Vorname Donald
  Nachname Duck
  Geburtsjahr 1934
```

```
7 }
8
9 parray Person
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_372.tcl
Person(Geburtsjahr) = 1934
Person(Nachname)    = Duck
Person(Vorname)     = Donald
oliver@debian:~$
```

Man kann ein Array auch anlegen, in dem man hinter `array set Variable` eine geschweifte Klammer `{` öffnet (Zeile 3) und dann die einzelnen Elemente des Arrays mit ihrem jeweiligen Wert aufführt (Zeilen 4 bis 6). Zum Schluss wird die geschweifte Klammer `}` geschlossen (Zeile 7).

Listing 13.3: Ausgabe verschiedener Array-Befehle (Beispiel373.tcl)

```
1  #!/usr/bin/env tclsh
2
3  array set Person {
4    Vorname Donald
5    Nachname Duck
6    Geburtsjahr 1934
7  }
8
9  puts "parray:"
10 parray Person
11 puts ""
12 puts "array get:"
13 puts [array get Person]
14 puts ""
15 puts "array names:"
16 puts [array names Person]
17 puts ""
18 puts "array size:"
19 puts [array size Person]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_373.tcl
parray:
Person(Geburtsjahr) = 1934
Person(Nachname)    = Duck
Person(Vorname)     = Donald

array get:
Vorname Donald Geburtsjahr 1934 Nachname Duck

array names:
Vorname Geburtsjahr Nachname

array size:
3
oliver@debian:~$
```

Listing 13.4: Element-Name wird über eine (numerische) Variable gebildet (Beispiel115.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Person(1,Vorname) Donald
4  set Person(1,Nachname) Duck
5  set Person(1,Geburtsjahr) 1934
6
7  set Person(2,Vorname) Mickey
8  set Person(2,Nachname) Mouse
9  set Person(2,Geburtsjahr) 1928
10
11 for {set i 1} {$i <=2} {incr i} {
12   puts "Person $i :"
13   puts "Vorname: $Person($i,Vorname)"
14   puts "Nachname: $Person($i,Nachname)"
15   puts "Geburtsjahr: $Person($i,Geburtsjahr)"
16 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_115.tcl
Person 1 :
Vorname: Donald
Nachname: Duck
Geburtsjahr: 1934
Person 2 :
Vorname: Mickey
Nachname: Mouse
Geburtsjahr: 1928
oliver@debian:~$
```

Listing 13.5: Element-Name wird über eine (alphabetische) Variable gebildet (Beispiel116.tcl)

```
#!/usr/bin/env tclsh

set Wochentag {Montag Dienstag Mittwoch Donnerstag Freitag}

set Sport(Montag) Fussball
set Sport(Dienstag) Badminton
set Sport(Mittwoch) Basketball
set Sport(Donnerstag) Volleyball
set Sport(Freitag) Handball

foreach Tag $Wochentag {
  puts "$Tag : $Sport($Tag)"
}
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_116.tcl
Montag : Fußball
Dienstag : Badminton
Mittwoch : Basketball
Donnerstag : Volleyball
Freitag : Handball
oliver@debian:~$
```

Listing 13.6: Alle Elemente eines Arrays durchlaufen (Beispiel117.tcl)

```
#!/usr/bin/env tclsh

set Sport(Montag) Fussball
set Sport(Dienstag) Badminton
set Sport(Mittwoch) Basketball
set Sport(Donnerstag) Volleyball
set Sport(Freitag) Handball

foreach Tag [array names Sport *] {
  puts "$Tag : $Sport($Tag)"
}
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_117.tcl
Donnerstag : Volleyball
Freitag : Handball
Montag : Fußball
Mittwoch : Basketball
Dienstag : Badminton
oliver@debian:~$
```

Das Beispiel ist fast wie Beispiel 116, aber die `foreach`-Schleife durchläuft automatisch alle Elemente des Arrays. Man erkennt außerdem, dass die Reihenfolge der Array-Elemente

nicht der Reihenfolge entspricht, wie das Array befüllt wurde. Die Elemente eines Arrays haben (im Unterschied zu Listen) keine bestimmte Index-Position.

Listing 13.7: Zweidimensionale Tabelle als Array (Beispiel118.tcl)

```
#!/usr/bin/env tclsh

set Wert(1,1) 101
set Wert(1,2) 102
set Wert(1,3) 103

set Wert(2,1) 201
set Wert(2,2) 202
set Wert(2,3) 203

for {set i 1} {$i <=2} {incr i} {
  for {set j 1} {$j <=3} {incr j} {
    puts "Zeile $i / Spalte $j : $Wert($i,$j)"
  }
}
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_118.tcl
Zeile 1 / Spalte 1 : 101
Zeile 1 / Spalte 2 : 102
Zeile 1 / Spalte 3 : 103
Zeile 2 / Spalte 1 : 201
Zeile 2 / Spalte 2 : 202
Zeile 2 / Spalte 3 : 203
oliver@debian:~$
```

Bei genauer Betrachtung handelt es sich in dem Beispiel nicht um numerische Indexwerte, sondern um textliche Element-Namen mit einem Komma in der Mitte. Der Element-Name ist nicht (Nummer,Nummer), wie man das aus anderen Programmiersprachen kennt, sondern der Element-Name ist ein Text, der aus den Zeichen 1 Komma 1 (1 Komma 2 usw.) besteht.

13.2 Array an Prozedur übergeben

Im Unterschied zu normalen Variablen, Listen und dem Dictionary (siehe Kapitel 14 auf Seite 161) kann man ein Array nicht direkt an eine Prozedur übergeben. Das liegt daran, wie der Inhalt des Arrays von Tcl/Tk gespeichert wird. Um ein Array in einer Prozedur zu verwenden, gibt es vier Möglichkeiten:

- man deklariert mit dem Befehl `global` das Array als globale Variable
- man verwendet den Befehl `upvar`
- man wandelt das Array in eine Liste um und übergibt die Liste
- man verwendet statt eines Arrays ein Dictionary

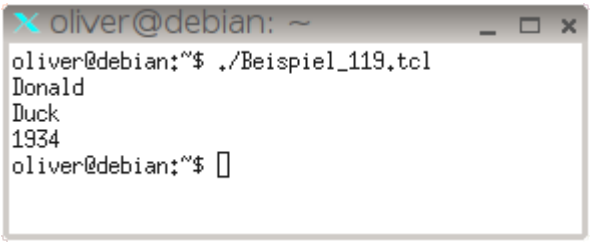
Der Befehl `global` ist bereits in einem früheren Kapital behandelt worden.

Der Befehl `upvar` erstellt einen Verweis auf das Array. Das funktioniert aber nur, wenn die Prozedur direkt vom Hauptprogramm aus aufgerufen wird. Wenn die Prozedur von einer anderen Prozedur aufgerufen wird (z. B. das Hauptprogramm ist Level 0, die erste Prozedur ist Level 1 und die Prozedur mit dem Array ist Level 2), muss man beim `upvar`-Befehl das Level angeben; in diesem Fall `upvar 2`.

Mit dem Befehl `set Liste [array get ArrayVariable]` kann man das Array in eine Liste umwandeln. Anschließend übergibt man die Liste an die Prozedur und wandelt in der Prozedur mit dem Befehl `set ArrayVariable [array set $Liste]` die Liste wieder in ein Array um. In diesem Fall ist die Array-Variable nur lokal innerhalb der Prozedur gültig.

Listing 13.8: Mit dem Befehl upvar auf ein Array zugreifen (Beispiel119.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc ArrayAnzeigen {tmp} {
4    upvar $tmp Person
5    puts $Person(Vorname)
6    puts $Person(Nachname)
7    puts $Person(Geburtsjahr)
8  }
9
10 set Person(Vorname) Donald
11 set Person(Nachname) Duck
12 set Person(Geburtsjahr) 1934
13 ArrayAnzeigen Person ; # ohne Dollarzeichen !!!
```



In Zeile 13 wird nur der Name des Arrays an die Prozedur übergeben. In Zeile 4 wird ein Verweis auf das Array erzeugt, so dass innerhalb der Prozedur auf den Inhalt des Arrays zugegriffen werden kann.

Listing 13.9: Ein Array mit upvar in einer Unter-Unter-Prozedur verwenden (Beispiel369.tcl)

```
#!/usr/bin/env tclsh

proc ArrayAnzeigen {tmp} {
  upvar 2 $tmp Person
  puts $Person(Vorname)
  puts $Person(Nachname)
  puts $Person(Geburtsjahr)
}
```

```
9
10 proc EineProzedur {} {
11         ArrayAnzeigen Person
12 }
13
14 set Person(Vorname) Donald
15 set Person(Nachname) Duck
16 set Person(Geburtsjahr) 1934
17 EineProzedur
```

In den Zeilen 14 bis 16 wird ein Array definiert. In Zeile 17 wird eine Prozedur aufgerufen, die ihrerseits die Prozedur `ArrayAnzeigen` aufruft. In Zeile 4 greift die Prozedur `ArrayAnzeigen` auf das Array zu. Da die Prozedur zwei Ebenen unterhalb des Hauptprogramms läuft, muss man dem `upvar`-Befehl mitteilen, dass das Array zwei Ebenen höher existiert. Dies macht man, indem man die Zahl 2 hinzufügt.

Wenn man in einer Prozedur mit dem `upvar`-Befehl auf das Array zugreift und das Array in der Prozedur verändert, wirkt sich das unmittelbar auf das Original-Array aus. Oft ist es besser, das Array innerhalb der Prozedur von dem Original-Array zu entkoppeln. Dazu wandelt man das Array in eine Liste um und übergibt die Liste an die Prozedur. Innerhalb der Prozedur wird die Liste in ein lokal gültiges Array umgewandelt. Man kann dadurch das (lokale) Array in der Prozedur verändern, ohne dass das Original-Array geändert wird. Bei Bedarf kann das geänderte Array von der Prozedur wieder an das Hauptprogramm zurückgegeben werden.

Listing 13.10: Ein Array mit Hilfe einer Liste an die Prozedur übergeben (Beispiel400.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc ArrayAnzeigen {LokaleListe} {
4    array set LokalePerson $LokaleListe
5    puts $LokalePerson(Vorname)
6    puts $LokalePerson(Nachname)
7    puts $LokalePerson(Geburtsjahr)
8  }
9
10 set Person(Vorname) Donald
11 set Person(Nachname) Duck
12 set Person(Geburtsjahr) 1934
13
14 set Liste [array get Person]
15 ArrayAnzeigen $Liste
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_400.tcl
Donald
Duck
1934
oliver@debian:~$
```

In Zeile 14 wird das Array Person mit dem Befehl `array get Person` in eine Liste umgewandelt. In Zeile 15 wird die Liste an die Prozedur übergeben. In Zeile 4 wird die Liste `LokaleListe` mit dem Befehl `array set $LokaleListe` wieder in ein Array umgewandelt.

Listing 13.11: Ein Array an eine Prozedur übergeben, lokal ändern und wieder zurückgeben (Beispiel511.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc ArrayAendern {Liste} {
4    array set Person $Liste
5    puts "In der Prozedur:"
6    parray Person
7    set Person(Vorname) Mickey
8    set Person(Nachname) Mouse
9    set Person(Geburtsjahr) 1928
10   return [array get Person]
11 }
12
13 set PersonAlt(Vorname) Donald
14 set PersonAlt(Nachname) Duck
15 set PersonAlt(Geburtsjahr) 1934
16
17 puts "Vorher:"
18 parray PersonAlt
19 puts "------------------"
20 array set PersonNeu [ArrayAendern [array get PersonAlt]]
21 puts "------------------"
22 puts "Nachher:"
23 parray PersonAlt
24 parray PersonNeu
```

```
Datei    Bearbeiten    Ansicht    >
oliver@debian:~$ ./Beispiel511.tc
Vorher:
PersonAlt(Geburtsjahr) = 1934
PersonAlt(Nachname)    = Duck
PersonAlt(Vorname)     = Donald
------------------
In der Prozedur:
Person(Geburtsjahr) = 1934
Person(Nachname)    = Duck
Person(Vorname)     = Donald
------------------
Nachher:
PersonAlt(Geburtsjahr) = 1934
PersonAlt(Nachname)    = Duck
PersonAlt(Vorname)     = Donald
PersonNeu(Geburtsjahr) = 1928
PersonNeu(Nachname)    = Mouse
PersonNeu(Vorname)     = Mickey
oliver@debian:~$
```

In den Zeilen 13 bis 15 wird das Array `PersonAlt` definiert. In Zeile 20 wird dieses Array als Liste an die Prozedur `ArrayAendern` übergeben. In der Prozedur wird in Zeile 4 die Liste in das lokal gültige Array `Person` überführt. In den Zeilen 7 bis 9 wird das lokale Array geändert, ohne dass das ursprüngliche Array `PersonAlt` geändert wird. In Zeile 10 wird das lokale Array wieder in eine Liste umgewandelt und an das Hauptprogramm zurückgegeben. In Zeile 20 wird die aus der Prozedur zurückgegebene Liste in das Array `PersonNeu` umgewandelt.

14 Dictionary

14.1 Dictionary

Ein Dictionary ist wie ein Wörterbuch. Man sucht nach einem Begriff (Schlüssel) und erhält den dazu passenden Wert. Das Dictionary speichert sogenannte Schlüssel-Wert-Paare.

Befehl 1: Einfaches Dictionary

```
set Variable {
 Schlüssel1 Wert1
 Schlüssel2 Wert2
 ...
}
puts [dict get $Variable Schlüssel1]
```

Befehl 2: Dictionary innerhalb eines Dictionary

```
set Variable {
 SchlüsselA {
 SchlüsselA1 WertA1
 SchlüsselA2 WertA2
 ...
 }
 SchlüsselB {
 SchlüsselB1 WertB1
 SchlüsselB2 WertB2
 ...
 }
}
puts [dict get [dict get $Variable SchlüsselA] SchlüsselA1]
puts [dict get $Variable SchlüsselA SchlüsselA1]
```

Man definiert ein Dictionary durch geschweifte Klammern `{}`, in die man Schlüssel-Wert-Paare setzt. Die Schlüssel müssen eindeutig sein.

Tabelle 14.1: Die wichtigsten Dictionary-Befehle

Befehl	Beschreibung
`dict set Person Vorname Donald`	Setzt bzw. ändert den Wert zum Schlüssel. Wenn es den Schlüssel noch nicht gibt, wird der Schlüssel hinzugefügt.

Tabelle 14.1: Die wichtigsten Dictionary-Befehle

Befehl	Beschreibung
`dict set Person 1 Vorname Donald`	Bei Dictionary in Dictionary: Setzt bzw. ändert den Wert zum Schlüssel. Wenn es den Schlüssel noch nicht gibt, wird der Schlüssel hinzugefügt.
`puts [dict get $Person Vorname]`	Gibt den Wert zum Schlüssel zurück
`puts [dict get $Person 1 Vorname]`	Bei Dictionary in Dictionary: Gibt den Wert zum Schlüssel zurück
`puts [dict size $Person]`	Anzahl der Schlüssel-Wert-Paare
`puts [dict keys $Person]`	Erzeugt eine Liste aller Schlüssel
`dict for {Schluessel Wert} $Person {puts "$Schluessel : $Wert"}`	Listet alle Schlüssel-Wert-Paare auf
`set Person [dict remove $Person Schlüssel]`	Löscht einen Eintrag im Dictionary
`set NeuesDict [dict merge $Person $Adresse]`	Fügt zwei Dictionaries zusammen
`dict append Person Name " Duck"`	Fügt dem zum Schlüssel gehörigen Wert einen Text hinzu.
`dict lappend Person Neffen Track`	Wenn der zum Schlüssel gehörige Wert eine Liste ist, kann man mit dem Befehl dict lappend der Liste ein weiteres Element hinzufügen.

Listing 14.1: Dictionary erzeugen (Beispiel120.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Person {
4    Vorname Donald
5    Nachname Duck
6    Geburtsjahr 1934
7  }
8
9  puts "Vorname: [dict get $Person Vorname]"
10 puts "Nachname: [dict get $Person Nachname]"
11 puts "Geburtsjahr: [dict get $Person Geburtsjahr]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_120.tcl
Vorname: Donald
Nachname: Duck
Geburtsjahr: 1934
oliver@debian:~$
```

Listing 14.2: Dictionary mit dict set erzeugen (Beispiel121.tcl)

```
#!/usr/bin/env tclsh

dict set Person Vorname Donald
dict set Person Nachname Duck
dict set Person Geburtsjahr 1934

puts "Vorname: [dict get $Person Vorname]"
puts "Nachname: [dict get $Person Nachname]"
puts "Geburtsjahr: [dict get $Person Geburtsjahr]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_121.tcl
Vorname: Donald
Nachname: Duck
Geburtsjahr: 1934
oliver@debian:~$
```

Listing 14.3: Dictionary in Dictionary erzeugen (Beispiel122.tcl)

```
#!/usr/bin/env tclsh

set Person {
  1 {
        Vorname Donald
        Nachname Duck
        Geburtsjahr 1934
  }
  2 {
        Vorname Mickey
        Nachname Mouse
        Geburtsjahr 1928
  }
}

puts "Anzahl Personen: [dict size $Person]"

puts "Person 1:"
puts "Vorname: [dict get [dict get $Person 1] Vorname]"
puts "Nachname: [dict get [dict get $Person 1] Nachname]"
```

```
21 puts "Geburtsjahr: [dict get [dict get $Person 1] ↩
   Geburtsjahr]"
22
23 puts "Person 2:"
24 puts "Vorname: [dict get $Person 2 Vorname]"
25 puts "Nachname: [dict get $Person 2 Nachname]"
26 puts "Geburtsjahr: [dict get $Person 2 Geburtsjahr]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_122.tcl
Anzahl Personen: 2
Person 1:
Vorname: Donald
Nachname: Duck
Geburtsjahr: 1934
Person 2:
Vorname: Mickey
Nachname: Mouse
Geburtsjahr: 1928
oliver@debian:~$
```

In Zeile 16 wird die Anzahl der Personen angezeigt. In den Zeilen 19 bis 21 und 24 bis 26 werden die Personendaten angezeigt. In den Zeilen 19 bis 21 werden die `dict get`-Befehle ineinander geschachtelt. Die innere Anweisung holt den gesamten Datensatz zur Person, die äußere Anweisung den eigentlichen Wert, z. B. den Vornamen. In den Zeilen 24 bis 26 ist die Kurzform dargestellt.

Listing 14.4: Dictionary in Dictionary mit dict set erzeugen (Beispiel123.tcl)

```
1 #!/usr/bin/env tclsh
2
3 dict set Person 1 Vorname Donald
4 dict set Person 1 Nachname Duck
5 dict set Person 1 Geburtsjahr 1934
6
7 dict set Person 2 Vorname Mickey
8 dict set Person 2 Nachname Mouse
9 dict set Person 2 Geburtsjahr 1928
10
11 puts "Anzahl Personen: [dict size $Person]"
12
13 puts "Person 1:"
14 puts "Vorname: [dict get [dict get $Person 1] Vorname]"
15 puts "Nachname: [dict get [dict get $Person 1] Nachname]"
16 puts "Geburtsjahr: [dict get [dict get $Person 1] ↩
   Geburtsjahr]"
17
18 puts "Person 2:"
19 puts "Vorname: [dict get $Person 2 Vorname]"
20 puts "Nachname: [dict get $Person 2 Nachname]"
21 puts "Geburtsjahr: [dict get $Person 2 Geburtsjahr]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_123.tcl
Anzahl Personen: 2
Person 1:
Vorname: Donald
Nachname: Duck
Geburtsjahr: 1934
Person 2:
Vorname: Mickey
Nachname: Mouse
Geburtsjahr: 1928
oliver@debian:~$
```

Listing 14.5: Dictionary mit Hilfe von Variablen erzeugen (Beispiel124.tcl)

```
#!/usr/bin/env tclsh

set VorName Donald
set NachName Duck
set GeburtsJahr 1934

dict set Person Vorname $VorName
dict set Person Nachname $NachName
dict set Person Geburtsjahr $GeburtsJahr

puts [dict get $Person]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_124.tcl
Vorname Donald Nachname Duck Geburtsjahr 1934
oliver@debian:~$
```

Listing 14.6: Alle Schlüssel-Wert-Paare mit foreach anzeigen (Beispiel125.tcl)

```
#!/usr/bin/env tclsh

dict set Person Vorname Donald
dict set Person Nachname Duck
dict set Person Geburtsjahr 1934

foreach Schluessel [dict keys $Person] {
  set Wert [dict get $Person $Schluessel]
  puts "Der Schluessel $Schluessel hat den Wert $Wert"
}
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_125.tcl
Der Schlüssel Vorname hat den Wert Donald
Der Schlüssel Nachname hat den Wert Duck
Der Schlüssel Geburtsjahr hat den Wert 1934
oliver@debian:~$
```

Listing 14.7: Alle Schlüssel-Wert-Paare mit dict for anzeigen (Beispiel126.tcl)

```
1 #!/usr/bin/env tclsh
2
3 dict set Person Vorname Donald
4 dict set Person Nachname Duck
5 dict set Person Geburtsjahr 1934
6
7 dict for {Schluessel Wert} $Person {
8   puts "Der Schluessel $Schluessel hat den Wert $Wert"
9 }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_126.tcl
Der Schlüssel Vorname hat den Wert Donald
Der Schlüssel Nachname hat den Wert Duck
Der Schlüssel Geburtsjahr hat den Wert 1934
oliver@debian:~$
```

Listing 14.8: Den Wert zu einem Schlüssel ändern (Beispiel127.tcl)

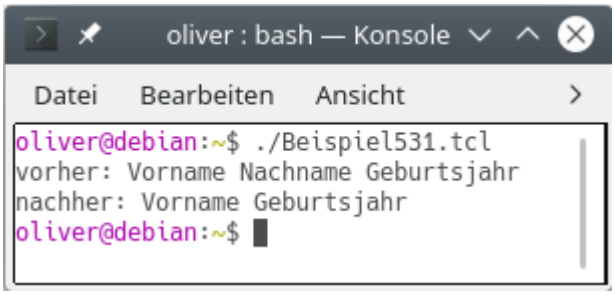
```
1 #!/usr/bin/env tclsh
2
3 dict set Person Vorname Donald
4 dict set Person Nachname Duck
5 dict set Person Geburtsjahr 1943
6
7 puts "Falsches Geburtsjahr: [dict get $Person Geburtsjahr]
    "
8 dict set Person Geburtsjahr 1934
9 puts "Korrigiertes Geburtsjahr: [dict get $Person
    Geburtsjahr]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_127.tcl
Falsches Geburtsjahr: 1943
Korrigiertes Geburtsjahr: 1934
oliver@debian:~$
```

In Zeile 8 wird das Geburtsjahr neu gesetzt.

Listing 14.9: Einen Schlüssel löschen (Beispiel531.tcl)

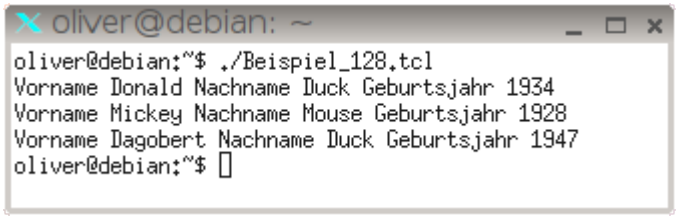
```
1 #!/usr/bin/env tclsh
2
3 dict set Person Vorname Donald
4 dict set Person Nachname Duck
5 dict set Person Geburtsjahr 1943
6
7 puts "vorher: [dict keys $Person]"
8 set Person [dict remove $Person Nachname]
9 puts "nachher: [dict keys $Person]"
```



In Zeile 8 wird der Schlüssel `Nachname` gelöscht.

Listing 14.10: Zu einem geschachtelten Dictionary einen Datensatz hinzufügen (Beispiel128.tcl)

```
#!/usr/bin/env tclsh

dict set Person 1 Vorname Donald
dict set Person 1 Nachname Duck
dict set Person 1 Geburtsjahr 1934

dict set Person 2 Vorname Mickey
dict set Person 2 Nachname Mouse
dict set Person 2 Geburtsjahr 1928

puts [dict get $Person 1]
puts [dict get $Person 2]

dict set Person 3 Vorname Dagobert
dict set Person 3 Nachname Duck
dict set Person 3 Geburtsjahr 1947

puts [dict get $Person 3]
```



In den Zeilen 14 bis 16 wird eine weitere Person zum Dictionary hinzugefügt.

Listing 14.11: Zwei Dictionaries zusammenfügen (Beispiel129.tcl)

```
1  #!/usr/bin/env tclsh
2
3  dict set Person Vorname Donald
4  dict set Person Nachname Duck
5  dict set Person Geburtsjahr 1934
6
7  dict set Adresse Ort Entenhausen
8  dict set Adresse Land USA
9
10 puts "Person: [dict get $Person]"
11 puts "Adresse: [dict get $Adresse]"
12
13 set Gesamt [dict merge $Person $Adresse]
14 puts "Gesamt: [dict get $Gesamt]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_129.tcl
Person: Vorname Donald Nachname Duck Geburtsjahr 1934
Adresse: Ort Entenhausen Land USA
Gesamt: Vorname Donald Nachname Duck Geburtsjahr 1934 Ort Entenhausen Land USA
oliver@debian:~$
```

In Zeile 13 werden die beiden Dictionaries Person und Adresse zusammengefasst.

Listing 14.12: Einen Wert im Dictionary ergänzen (Beispiel130.tcl)

```
1  #!/usr/bin/env tclsh
2
3  dict set Person Name Donald
4  dict set Person Ort Entenhausen
5  puts "vorher: [dict get $Person]"
6  dict append Person Name " Duck"
7  puts "nachher: [dict get $Person]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_130.tcl
vorher: Name Donald Ort Entenhausen
nachher: Name {Donald Duck} Ort Entenhausen
oliver@debian:~$
```

In Zeile 6 wird der Name ergänzt.

Listing 14.13: Einen Wert (Liste) im Dictionary ergänzen (Beispiel131.tcl)

```
1  #!/usr/bin/env tclsh
2
```

```
3  set ListeNeffen {}
4  lappend ListeNeffen Tick
5  lappend ListeNeffen Trick
6
7  dict set Person Name Donald
8  dict set Person Neffen $ListeNeffen
9
10 puts "vorher: [dict get $Person]"
11 set Liste [dict get $Person Neffen]
12 puts "Anzahl Neffen: [llength $Liste] Personen"
13
14 dict lappend Person Neffen Track
15
16 puts "nachher: [dict get $Person]"
17 set Liste [dict get $Person Neffen]
18 puts "Anzahl Neffen: [llength $Liste] Personen"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_131.tcl
vorher: Name Donald Neffen {Tick Trick}
Anzahl Neffen: 2 Personen
nachher: Name Donald Neffen {Tick Trick Track}
Anzahl Neffen: 3 Personen
oliver@debian:~$
```

In Zeile 14 wird eine weitere Person zu der Liste mit den Neffen hinzugefügt.

14.2 Unterschied zwischen Array und Dictionary

Ein Array hat gegenüber dem Dictionary in manchen Situationen ein paar Nachteile:

- In einem Array kann man kein weiteres Array speichern.
- Man kann ein Array nicht direkt an eine Prozedur übergeben, sondern muss den Befehl `upvar` verwenden.

Dem gegenüber kann man ein Dictionary in einem anderen Dictionary speichern und außerdem kann man ein Dictionary direkt an eine Prozedur übergeben.

15 Stack (Stapel)

Ein Stack ist wie ein Stapel auf dem man Elemente platziert. Das erste auf dem Stapel abgelegte Element liegt an unterster Stelle, das nächste Element darüber. Wenn man ein Element vom Stapel herunter holt, wird immer das oberste (zuletzt auf den Stapel gelegte) Elemente genommen.

Je nachdem wie man den Stapel erzeugt, werden die Stack-Befehle verwendet. In der ersten Variante wird der Stapel mit dem Befehl `::struct::stack MeinStapel` erzeugt. In Folge dessen wird ein globales Objekt namens `MeinStapel` erstellt, so dass alle Stack-Befehle mit `MeinStapel` beginnen (z. B. `MeinStapel size`). In der zweiten Variante wird der Stapel mit dem Befehl `set MeinStapel [::struct::stack]` erzeugt. Dadurch wird das Stack-Objekt in der Variablen `MeinStapel` gespeichert. Die Stack-Befehle beginnen deshalb mit `$MeinStapel` (z. B. `$MeinStapel size`).

Tabelle 15.1: Die wichtigsten Befehle für den Stack (gemäß Variante 1)

Befehl	Beschreibung
`package require struct::stack`	Bindet das Stack-Paket in das Programm ein
`::struct::stack MeinStapel`	Erzeugt einen leeren Stapel (Variante 1)
`MeinStapel size`	Ermittelt die Anzahl der Elemente
`MeinStapel push Element`	Legt ein Element auf den Stapel
`MeinStapel push {*}$Liste`	Legt die Elemente einer Liste einzeln auf den Stapel
`MeinStapel pop Anzahl`	Holt eine Anzahl an Elementen vom Stapel. Wenn `Anzahl` nicht angegeben wird, wird genau ein Element geholt. Wenn mehrere Elemente geholt werden, ist das Ergebnis eine Liste.
`MeinStapel get`	Holt alle Elemente vom Stapel, ohne die Elemente vom Stapel zu entfernen.
`MeinStapel getr`	Holt alle Elemente in umgekehrter Reihenfolge vom Stapel, ohne die Elemente vom Stapel zu entfernen.

Tabelle 15.1: Die wichtigsten Befehle für den Stack (gemäß Variante 1)

Befehl	Beschreibung
`MeinStapel peek Anzahl`	Holt eine Anzahl an Elementen vom Stapel, ohne die Elemente vom Stapel zu entfernen. Wenn `Anzahl` nicht angegeben wird, wird genau ein Element geholt.
`MeinStapel peekr Anzahl`	Holt eine Anzahl an Elementen in umgekehrter Reihenfolge vom Stapel, ohne die Elemente vom Stapel zu entfernen. Wenn `Anzahl` nicht angegeben wird, wird genau ein Element geholt.
`MeinStapel clear`	Entfernt alle Elemente vom Stapel
`MeinStapel destroy`	Löscht das Stapel-Objekt. Es existiert danach nicht mehr.

Tabelle 15.2: Die wichtigsten Befehle für den Stack (gemäß Variante 2)

Befehl	Beschreibung
`package require struct::stack`	Bindet das Stack-Paket in das Programm ein
`set MeinStapel ::struct::stack`	Erzeugt einen leeren Stapel (Variante 2)
`$MeinStapel size`	Ermittelt die Anzahl der Elemente
`$MeinStapel push Element`	Legt ein Element auf den Stapel
`$MeinStapel push {*}$Liste`	Legt die Elemente einer Liste einzeln auf den Stapel
`$MeinStapel pop Anzahl`	Holt eine Anzahl an Elementen vom Stapel. Wenn `Anzahl` nicht angegeben wird, wird genau ein Element geholt. Wenn mehrere Elemente geholt werden, ist das Ergebnis eine Liste.
`$MeinStapel get`	Holt alle Elemente vom Stapel, ohne die Elemente vom Stapel zu entfernen.
`$MeinStapel getr`	Holt alle Elemente in umgekehrter Reihenfolge vom Stapel, ohne die Elemente vom Stapel zu entfernen.

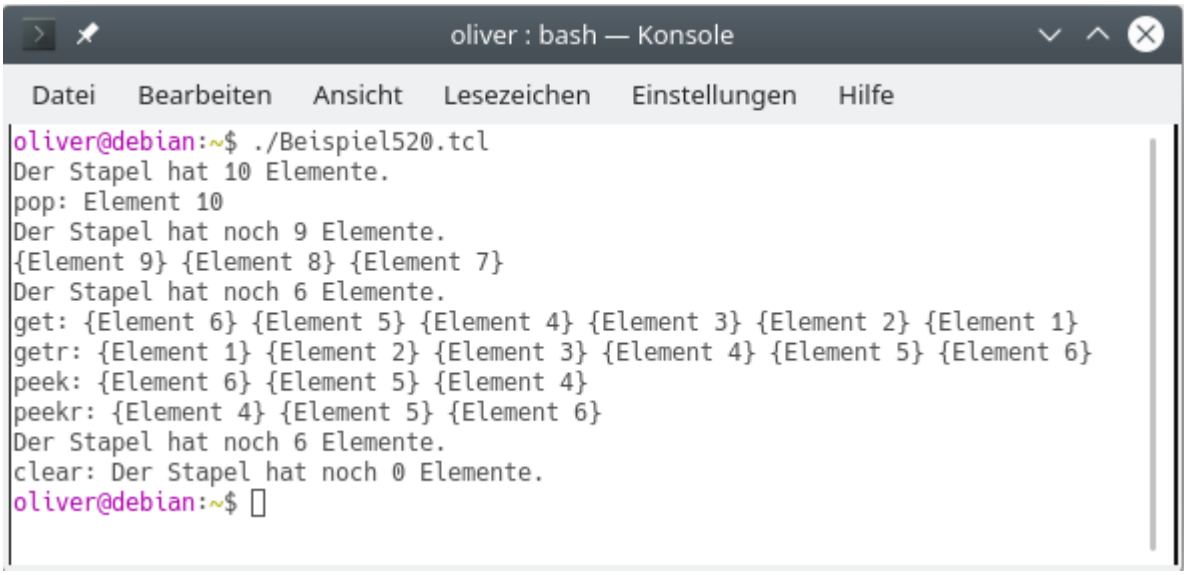
Tabelle 15.2: Die wichtigsten Befehle für den Stack (gemäß Variante 2)

Befehl	Beschreibung
`$MeinStapel peek Anzahl`	Holt eine Anzahl an Elementen vom Stapel, ohne die Elemente vom Stapel zu entfernen. Wenn `Anzahl` nicht angegeben wird, wird genau ein Element geholt.
`$MeinStapel peekr Anzahl`	Holt eine Anzahl an Elementen in umgekehrter Reihenfolge vom Stapel, ohne die Elemente vom Stapel zu entfernen. Wenn `Anzahl` nicht angegeben wird, wird genau ein Element geholt.
`$MeinStapel clear`	Entfernt alle Elemente vom Stapel
`$MeinStapel destroy`	Löscht das Stapel-Objekt. Es existiert danach nicht mehr.

Listing 15.1: Stack (Variante 1) (Beispiel520.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require struct::stack
4
5  ::struct::stack MeinStapel
6
7  for {set i 1} {$i <= 7} {incr i} {
8    MeinStapel push "Element $i"
9  }
10 set Liste {"Element 8" "Element 9" "Element 10"}
11 MeinStapel push {*}$Liste
12
13 puts "Der Stapel hat [MeinStapel size] Elemente."
14
15 set Element [MeinStapel pop]
16 puts "pop: $Element"
17 puts "Der Stapel hat noch [MeinStapel size] Elemente."
18
19 set Liste [MeinStapel pop 3]
20 puts $Liste
21 puts "Der Stapel hat noch [MeinStapel size] Elemente."
22
23 set Liste [MeinStapel get]
24 puts "get: $Liste"
25
26 set Liste [MeinStapel getr]
27 puts "getr: $Liste"
28
29 set Liste [MeinStapel peek 3]
30 puts "peek: $Liste"
```

```
31
32 set Liste [MeinStapel peekr 3]
33 puts "peekr: $Liste"
34
35 puts "Der Stapel hat noch [MeinStapel size] Elemente."
36
37 MeinStapel clear
38 puts "clear: Der Stapel hat noch [MeinStapel size] ↩
   Elemente."
39
40 MeinStapel destroy
```



Listing 15.2: Stack (Variante 2) (Beispiel521.tcl)

```
1 #!/usr/bin/env tclsh
2
3 package require struct::stack
4
5 set MeinStapel [::struct::stack]
6
7 for {set i 1} {$i <= 7} {incr i} {
8   $MeinStapel push "Element $i"
9 }
10 set Liste {"Element 8" "Element 9" "Element 10"}
11 $MeinStapel push {*}$Liste
12
13 puts "Der Stapel hat [$MeinStapel size] Elemente."
14
15 set Element [$MeinStapel pop]
16 puts "pop: $Element"
17 puts "Der Stapel hat noch [$MeinStapel size] Elemente."
18
```

```
19 set Liste [$MeinStapel pop 3]
20 puts $Liste
21 puts "Der Stapel hat noch [$MeinStapel size] Elemente."
22
23 set Liste [$MeinStapel get]
24 puts "get: $Liste"
25
26 set Liste [$MeinStapel getr]
27 puts "getr: $Liste"
28
29 set Liste [$MeinStapel peek 3]
30 puts "peek: $Liste"
31
32 set Liste [$MeinStapel peekr 3]
33 puts "peekr: $Liste"
34
35 puts "Der Stapel hat noch [$MeinStapel size] Elemente."
36
37 $MeinStapel clear
38 puts "clear: Der Stapel hat noch [$MeinStapel size] ↲
     Elemente."
39
40 $MeinStapel destroy
```

```
oliver : bash — Konsole
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
oliver@debian:~$ ./Beispiel521.tcl
Der Stapel hat 10 Elemente.
pop: Element 10
Der Stapel hat noch 9 Elemente.
{Element 9} {Element 8} {Element 7}
Der Stapel hat noch 6 Elemente.
get: {Element 6} {Element 5} {Element 4} {Element 3} {Element 2} {Element 1}
getr: {Element 1} {Element 2} {Element 3} {Element 4} {Element 5} {Element 6}
peek: {Element 6} {Element 5} {Element 4}
peekr: {Element 4} {Element 5} {Element 6}
Der Stapel hat noch 6 Elemente.
clear: Der Stapel hat noch 0 Elemente.
oliver@debian:~$
```

In Zeile 3 wird das Stack-Paket eingebunden. In Zeile 5 wird ein Stapel-Objekt erzeugt. In den Zeilen 7 bis 9 werden sieben Elemente auf den Stapel gelegt. In Zeile 11 werden drei weitere Elemente aus einer Liste auf den Stapel gelegt. Zeile 13 zeigt die Anzahl der Elemente auf dem Stapel an. In Zeile 15 wird das oberste Element des Stapels geholt. In Zeile 19 werden drei weitere Elemente vom Stapel geholt. In Zeile 23 wird der gesamte Stapel in eine Liste überführt. In Zeile 26 ebenfalls, allerdings in umgekehrter Reihenfolge. In Zeile 29 werden drei Elemente vom Stapel geholt, ohne dass sie vom Stapel gelöscht

werden. In Zeile 32 ebenfalls, aber in umgekehrter Reihenfolge. In Zeile 37 werden alle Elemente vom Stapel entfernt. In Zeile 40 wird das Stapel-Objekt gelöscht. Es existiert danach nicht mehr.

16 Pool

Ein Pool ist eine Menge von Einzelobjekten wie zum Beispiel die Sitzplätze im Kino oder der Fahrzeugbestand eines Autovermieters.

Je nachdem wie man den Pool erzeugt, werden die Pool-Befehle verwendet. In der ersten Variante wird der Pool mit dem Befehl `::struct::pool MeinPool` erzeugt. In Folge dessen wird ein globales Objekt namens `MeinPool` erstellt, so dass alle Pool-Befehle mit `MeinPool` beginnen (z. B. `MeinPool info allitems`). In der zweiten Variante wird der Pool mit dem Befehl `set MeinPool [::struct::pool]` erzeugt. Dadurch wird das Pool-Objekt in der Variablen `MeinPool` gespeichert. Die Pool-Befehle beginnen deshalb mit `$MeinPool` (z. B. `$MeinPool info allitems`).

Tabelle 16.1: Die wichtigsten Befehle für den Pool (gemäß Variante 1)

Befehl	Beschreibung
`package require struct::pool`	Bindet das Pool-Paket in das Programm ein
`::struct::pool MeinPool`	Erzeugt einen leeren Pool (Variante 1)
`MeinPool maxsize Wert`	Legt die maximale Anzahl der Elemente im Pool fest
`MeinPool add Element`	Fügt ein Element dem Pool hinzu
`MeinPool add Element1 Element2`	Fügt zwei Elemente dem Pool hinzu
`MeinPool add {*}$Liste`	Fügt die Elemente einer Liste dem Pool hinzu
`MeinPool remove Element`	Entfernt ein Element aus dem Pool
`MeinPool clear -force`	Entfernt alle Elemente aus dem Pool
`MeinPool destroy -force`	Löscht den Pool. Er existiert danach nicht mehr.
`MeinPool request Element -allocID Reservierer`	Reserviert ein beliebiges Element aus dem Pool (z. B. Sitzplatz Nr. 12) für den Reservierer (z. B. die Person Anton). Das reservierte Element wird in der Variable `Element` gespeichert.

Tabelle 16.1: Die wichtigsten Befehle für den Pool (gemäß Variante 1)

Befehl	Beschreibung
`MeinPool request Element -allocID Bezeichner -prefer WunschElement`	Reserviert ein bestimmtes Element aus dem Pool (z. B. Sitzplatz Nr. 12) für den Bezeichner (z. B. die Person Anton). Das reservierte Element wird in der Variable `Element` gespeichert. Wenn das Wunschelement nicht verfügbar ist, wird der Wert `0` zurückgegeben.
`MeinPool release Element`	Gibt ein reserviertes Element aus dem Pool wieder frei
`[MeinPool info allitems]`	Erstellt eine Liste aller Elemente des Pools
`[MeinPool info freeitems]`	Erstellt eine Liste aller freien Elemente des Pools
`[MeinPool info allocstate]`	Erstellt eine Liste (bestehend aus den Schlüssel-Wert-Paaren Element und Reservierer) mit allen Elementen des Pools. Freie Elemente zeigen als Reservierer `-1` an.
`[MeinPool info allocID Element]`	Gibt den Reservierer des Elements an. Wenn das Element frei ist, wird als Reservierer `-1` angezeigt.
`[MeinPool info cursize]`	Anzahl der Elemente im Pool
`[MeinPool info maxsize]`	Maximale Anzahl der Elemente im Pool

Listing 16.1: Pool (Variante 1) (Beispiel555.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require struct::pool
4
5  ::struct::pool Autopool
6  Autopool maxsize 20
7
8  Autopool add BMW Citroen Opel Fiat Honda Mazda
9  puts "Der Pool besteht aus:"
10 puts [Autopool info allitems]
11 puts ""
12
13 puts "Honda entfernen und Honda1 und Honda2 hinzufuegen:"
14 Autopool remove Honda
15 Autopool add Honda1 Honda2
16 puts [Autopool info allitems]
```

```
17 puts ""
18
19 puts "Anton bucht ein Auto:"
20 if {[Autopool request Auto -allocID "Anton"]} {
21         puts "$Auto wurde gebucht von [Autopool info ↩
           allocID $Auto]."
22 } else {
23         puts "Kein Auto verfuegbar."
24 }
25 puts [Autopool info allocstate]
26 puts ""
27
28 puts "Berta bucht ein Auto:"
29 if {[Autopool request Auto -allocID "Berta"]} {
30         puts "$Auto wurde gebucht von [Autopool info ↩
           allocID $Auto]."
31 } else {
32         puts "Kein Auto verfuegbar."
33 }
34 puts [Autopool info allocstate]
35 puts ""
36
37 puts "Dora bucht ein Auto und bevorzugt einen BMW:"
38 if {[Autopool request Auto -allocID "Dora" -prefer BMW]} {
39         puts "$Auto wurde gebucht von [Autopool info ↩
           allocID $Auto]."
40 } else {
41         puts "Der BMW ist nicht verfuegbar. Folgende Autos↩
            sind noch frei:"
42         puts [Autopool info freeitems]
43 }
44 puts ""
45
46 puts "Anton gibt das Auto zurueck:"
47 Autopool release BMW
48 puts [Autopool info allocstate]
49 puts ""
```

```
oliver@ubuntu: ~
oliver@ubuntu:~$ ./Beispiel555.tcl
Der Pool besteht aus:
Fiat Opel Citroen Mazda Honda BMW

Honda entfernen und Honda1 und Honda2 hinzufuegen:
Honda2 Fiat Opel Citroen Mazda BMW Honda1

Anton bucht ein Auto:
BMW wurde gebucht von Anton.
Honda2 -1 Fiat -1 Opel -1 Citroen -1 Mazda -1 BMW Anton Honda1 -1

Berta bucht ein Auto:
Citroen wurde gebucht von Berta.
Honda2 -1 Fiat -1 Opel -1 Citroen Berta Mazda -1 BMW Anton Honda1 -1

Dora bucht ein Auto und bevorzugt einen BMW:
Der BMW ist nicht verfuegbar. Folgende Autos sind noch frei:
Opel Fiat Mazda Honda1 Honda2

Anton gibt das Auto zurueck:
Honda2 -1 Fiat -1 Opel -1 Citroen Berta Mazda -1 BMW -1 Honda1 -1

oliver@ubuntu:~$
```

Listing 16.2: Pool (Variante 2) (Beispiel558.tcl)

```
#!/usr/bin/env tclsh

package require struct::pool

set Autopool [::struct::pool]
$Autopool maxsize 20

$Autopool add BMW Citroen Opel Fiat Honda Mazda
puts "Der Pool besteht aus:"
puts [$Autopool info allitems]
puts ""

puts "Honda entfernen und Honda1 und Honda2 hinzufuegen:"
$Autopool remove Honda
$Autopool add Honda1 Honda2
puts [$Autopool info allitems]
puts ""

puts "Anton bucht ein Auto:"
if {[$Autopool request Auto -allocID "Anton"]} {
        puts "$Auto wurde gebucht von [$Autopool info allocID $Auto]."
} else {
        puts "Kein Auto verfuegbar."
}
puts [$Autopool info allocstate]
puts ""

```

```
28 puts "Berta bucht ein Auto:"
29 if {[$Autopool request Auto -allocID "Berta"]} {
30         puts "$Auto wurde gebucht von [$Autopool info ↩
           allocID $Auto]."
31 } else {
32         puts "Kein Auto verfuegbar."
33 }
34 puts [$Autopool info allocstate]
35 puts ""
36
37 puts "Dora bucht ein Auto und bevorzugt einen BMW:"
38 if {[$Autopool request Auto -allocID "Dora" -prefer BMW]} ↩
   {
39         puts "$Auto wurde gebucht von [$Autopool info ↩
           allocID $Auto]."
40 } else {
41         puts "Der BMW ist nicht verfuegbar. Folgende Autos↩
            sind noch frei:"
42         puts [$Autopool info freeitems]
43 }
44 puts ""
45
46 puts "Anton gibt das Auto zurueck:"
47 $Autopool release BMW
48 puts [$Autopool info allocstate]
49 puts ""
```

```
oliver@ubuntu: ~
oliver@ubuntu:~$ ./Beispiel558.tcl
Der Pool besteht aus:
Fiat Opel Citroen Mazda Honda BMW

Honda entfernen und Honda1 und Honda2 hinzufuegen:
Honda2 Fiat Opel Citroen Mazda BMW Honda1

Anton bucht ein Auto:
BMW wurde gebucht von Anton.
Honda2 -1 Fiat -1 Opel -1 Citroen -1 Mazda -1 BMW Anton Honda1 -1

Berta bucht ein Auto:
Citroen wurde gebucht von Berta.
Honda2 -1 Fiat -1 Opel -1 Citroen Berta Mazda -1 BMW Anton Honda1 -1

Dora bucht ein Auto und bevorzugt einen BMW:
Der BMW ist nicht verfuegbar. Folgende Autos sind noch frei:
Opel Fiat Mazda Honda1 Honda2

Anton gibt das Auto zurueck:
Honda2 -1 Fiat -1 Opel -1 Citroen Berta Mazda -1 BMW -1 Honda1 -1

oliver@ubuntu:~$
```

In Zeile 3 wird das Pool-Paket eingebunden. In Zeile 5 wird ein Pool-Objekt erzeugt. In

Zeile 6 wird die maximale Größe des Pool auf 20 Elemente festgelegt. In Zeile 8 werden dem Pool sechs Elemente hinzugefügt. In Zeile 10 werden alle Elemente des Pools angezeigt. In Zeile 14 wird ein Element aus dem Pool entfernt und in Zeile 15 werden zwei neue Elemente dem Pool hinzugefügt. In den Zeilen 20 und 29 wird jeweils ein beliebiges Element reserviert. In der Zeile 25 werden alle Elemente mit ihrem Reservierer angezeigt. Wenn ein Element frei ist, wird als Reserviert `-1` angezeigt. In Zeile 38 soll ein bestimmtes Element reserviert werden. Das Element ist aber bereits reserviert, so dass in Zeile 42 alle freien Elemente angezeigt werden. In Zeile 47 wird die Reservierung für ein Element aufgeboben.

17 Tree (Baum)

Zum Speichern hierarchischer Strukturen (z. B. die Abbildung von Ordnern, Unterordnern und Dateien wie in einem Dateimanager) eignet sich ein `tree` (Baum). Ein Baum startet mit einem Wurzelknoten (Elternknoten), unterhalb dessen weitere Knoten (Kinder) eingefügt werden. Die Knoten können weitere Knoten enthalten, so dass sie ihrerseits Eltern sind und Kinder enthalten. Hat ein Knoten keine Kinder, nennt man ihn ein Blatt.

Je nachdem wie man den Baum erzeugt, werden die Tree-Befehle verwendet. In der ersten Variante wird der Baum mit dem Befehl `::struct::tree MeinBaum` erzeugt. In Folge dessen wird ein globales Objekt namens `MeinBaum` erstellt, so dass alle Tree-Befehle mit `MeinBaum` beginnen (z. B. `MeinBaum children`). In der zweiten Variante wird der Stapel mit dem Befehl `set MeinBaum [::struct::tree]` erzeugt. Dadurch wird das Tree-Objekt in der Variablen `MeinBaum` gespeichert. Die Tree-Befehle beginnen deshalb mit `$MeinBaum` (z. B. `$MeinBaum children`).
Die Elemente im Baum werden über einen Index angesprochen. Wie z. B. auch bei Listen bezeichnet `end` den letzten Index und `end-1` den vorletzten usw.

Tabelle 17.1: Die wichtigsten Befehle für den Tree

Befehl	Beschreibung
`package require struct::tree`	Bindet das Tree-Paket in das Programm ein
`::struct::tree Baum`	Erzeugt einen leeren Baum
`Baum insert root end Index`	Fügt dem Wurzelknoten `root` einen Eintrag hinzu. Der Eintrag ist anhand des Index ansprechbar.
`Baum set Index Schluessel Wert`	Fügt dem Element mit dem Index `Index` ein Schlüssel-Wert-Paar (analog dem Dictionary) hinzu.
`Baum insert Index1 end Index2`	Fügt dem Element mit dem Index `Index1` einen Eintrag hinzu. Der Eintrag ist anhand des Index `Index2` ansprechbar.
`Baum children -all root`	Zeigt alle Kinder, Enkelkinder usw. des Wurzelknotens `root` an. Die Rückgabe sind immer die Indices der Kinder.

Tabelle 17.1: Die wichtigsten Befehle für den Tree

Befehl	Beschreibung
`Baum children root`	Zeigt die unmittelbaren Kinder des Wurzelknotens `root` an (ohne Enkelkinder usw.). Die Rückgabe sind immer die Indices der Kinder.
`Baum children -all Index`	Zeigt alle Kinder, Enkelkinder usw. des Elements mit dem Index `Index` an. Die Rückgabe sind immer die Indices der Kinder.
`Baum children Index`	Zeigt die unmittelbaren Kinder des Elements mit dem Index `Index` an (ohne Enkelkinder usw.). Die Rückgabe sind immer die Indices der Kinder.
`Baum numchildren Index`	Zeigt die Anzahl der Kinder des Elements mit dem Index `Index`.
`Baum parent Index`	Zeigt den Elternknoten des Elements mit dem Index `Index` an. Die Rückgabe ist der Index des Elternknotens.
`[Baum keys Index]`	Zeigt alle Schlüssel an, über die das Element mit dem Index `Index` verfügt.
`[Baum get Index Schluessel]`	Ermittelt den Wert des Schlüssels `Schluessel` des Elements mit dem Index `Index`.
`[Baum depth Index]`	Ermittelt die Entfernung des Elements mit dem Index `Index` vom Wurzelknoten.
`[Baum index Index]`	Ermittelt den Index des Elements mit dem Index `Index` innerhalb seines Elternknotens. Der erste Eintrag innerhalb des Elternknotens hat den Index 0.
`[Baum next Index]`	Ausgehend vom Element mit dem Index `Index` wird der Index des nächsten Elements innerhalb des Elternknotens ermittelt. Die Rückgabe ist leer, wenn es kein nächstes Element gibt.

Tabelle 17.1: Die wichtigsten Befehle für den Tree

Befehl	Beschreibung
`[Baum prevoius Index]`	Ausgehend vom Element mit dem Index `Index` wird der Index des vorherigen Elements innerhalb des Elternknotens ermittelt. Die Rückgabe ist leer, wenn es kein vorheriges Element gibt.
`[Baum keyexists Index Schluessel]`	Prüft, ob der Schlüssel `Schluessel` bei dem Element mit dem Index `Index` existiert (1 = ja, 0 = nein).
`Baum move ElternIndex end Index`	Verschiebt das Element mit dem Index `Index` in das Element mit dem Index `ElternIndex`. Das Element wird an der letzten Stelle eingefügt.
`Baum move ElternIndex StellenIndex Index`	Verschiebt das Element mit dem Index `Index` in das Element mit dem Index `ElternIndex`. Das Element wird an der Stelle `StellenIndex` eingefügt.
`Baum delete Index`	Löscht das Element mit dem Index `Index` inklusive aller Kinder.
`Baum cut Index`	Schneidet das Element mit dem Index `Index` aus. Die Kinder werden werden zu Kindern des Elterknotens des ausgeschnittenen Elements.
`Baum swap Index1 Index2`	Das Element mit dem Index `Index1` wird mit dem Element mit dem Index `Index2` vertauscht. Die Kinder sind davon nicht betroffen.

`Baum walk root -order pre -type bfs IndexVariable { Befehle}`	Durchwandert den Baum, beginnend beim Wurzelknoten `root` und führt für jedes Element die genannten Befehle aus. Der Index des aktuellen Elements wird in der Variable `IndexVariable` gespeichert und kann innerhalb der Befehle genutzt werden. Die Optionen `-order` und `-type` legt die Art und Weise fest, wie der Baum durchwandert wird:
	`-order pre`
	`-order post`
	`-type bfs`
	`-type dfs`

Tabelle 17.1: Die wichtigsten Befehle für den Tree

Befehl	Beschreibung

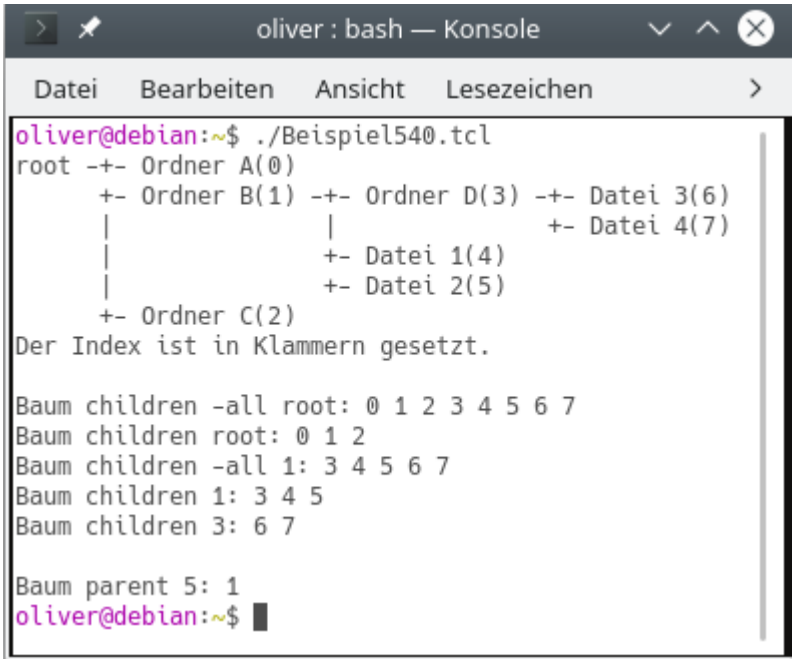
`Baum walk Index -order pre -type bfs IndexVariable {Befehle}`	Durchwandert den Baum, beginnend beim Element mit dem Index `Index` und führt für jedes Element die genannten Befehle aus. Der Index des aktuellen Elements wird in der Variable `IndexVariable` gespeichert und kann innerhalb der Befehle genutzt werden. Die Optionen `-order` und `-type` legt die Art und Weise fest, wie der Baum durchwandert wird:
	`-order pre`
	`-order post`
	`-type bfs`
	`-type dfs`

`Baum isleaf Index`	Prüft, ob das Element mit dem Index `Index` ein Blatt ist, d.h. es hat keine Unterelemente (1 = ja, 0 = nein).
`Baum leaves`	Erstellt eine Liste aller Blätter, also aller Elemente, die keine Unterelemente haben.
`Baum exists Index`	Prüft, ob das Element mit dem Index `Index` existiert (1 = ja, 0 = nein).

Listing 17.1: Tree erzeugen (Beispiel540.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require struct::tree
4
5  ::struct::tree Baum
6
7  # Es wird folgender Baum erfasst:
8  puts "root -+- Ordner A(0)"
9  puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
10 puts "      |               |                +- Datei 4(7)"
11 puts "      |               +- Datei 1(4)"
12 puts "      |               +- Datei 2(5)"
13 puts "      +- Ordner C(2)"
14 puts "Der Index ist in Klammern gesetzt."
15 puts ""
16
17 # Unterhalb von root:
18 Baum insert root end 0
19 Baum set 0 Name "Ordner A"
20 Baum set 0 Typ "Ordner"
21
```

```
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
24 Baum set 1 Typ "Ordner"
25
26 Baum insert root end 2
27 Baum set 2 Name "Ordner C"
28 Baum set 2 Typ "Ordner"
29
30 # Unterhalb von Ordner B:
31 Baum insert 1 end 3
32 Baum set 3 Name "Ordner D"
33 Baum set 3 Typ "Ordner"
34
35 Baum insert 1 end 4
36 Baum set 4 Name "Datei 1"
37 Baum set 4 Typ "Datei"
38
39 Baum insert 1 end 5
40 Baum set 5 Name "Datei 2"
41 Baum set 5 Typ "Datei"
42
43 # Unterhalb von Ordner D:
44 Baum insert 3 end 6
45 Baum set 6 Name "Datei 3"
46 Baum set 6 Typ "Datei"
47
48 Baum insert 3 end 7
49 Baum set 7 Name "Datei 4"
50 Baum set 7 Typ "Datei"
51
52 puts "Baum children -all root: [Baum children -all root]"
53 puts "Baum children root: [Baum children root]"
54 puts "Baum children -all 1: [Baum children -all 1]"
55 puts "Baum children 1: [Baum children 1]"
56 puts "Baum children 3: [Baum children 3]"
57 puts ""
58 puts "Baum parent 5: [Baum parent 5]"
```



In Zeile 3 wird das Paket `struct::tree` eingebunden. In Zeile 5 wird ein Tree mit dem Namen `Baum` erstellt. In den Zeilen 18 bis 20 wird unterhalb des Wurzelknotens `root` ein Element mit dem Index 0 erzeugt. Das Element hat die beiden Schlüssel `Name` und `Typ`. In den Zeilen 31 bis 33 wird unterhalb des Elements 1 (= Ordner B) ein Element mit dem Index 3 erstellt. In den Zeilen 52 bis 58 werden zu einzelnen Elementen die Kinder- und Eltern-Elemente abgefragt.

Listing 17.2: Tree leeren (Beispiel549.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require struct::tree
4
5  ::struct::tree Baum
6
7  # Es wird folgender Baum erfasst:
8  puts "root -+- Ordner A(0)"
9  puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
10 puts "      |               |                +- Datei 4(7)"
11 puts "      |               +- Datei 1(4)"
12 puts "      |               +- Datei 2(5)"
13 puts "      +- Ordner C(2)"
14 puts "Der Index ist in Klammern gesetzt."
15 puts ""
16
17 # Unterhalb von root:
18 Baum insert root end 0
19 Baum set 0 Name "Ordner A"
20 Baum set 0 Typ "Ordner"
21
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
```

```
24 Baum set 1 Typ "Ordner"
25
26 Baum insert root end 2
27 Baum set 2 Name "Ordner C"
28 Baum set 2 Typ "Ordner"
29
30 # Unterhalb von Ordner B:
31 Baum insert 1 end 3
32 Baum set 3 Name "Ordner D"
33 Baum set 3 Typ "Ordner"
34
35 Baum insert 1 end 4
36 Baum set 4 Name "Datei 1"
37 Baum set 4 Typ "Datei"
38
39 Baum insert 1 end 5
40 Baum set 5 Name "Datei 2"
41 Baum set 5 Typ "Datei"
42
43 # Unterhalb von Ordner D:
44 Baum insert 3 end 6
45 Baum set 6 Name "Datei 3"
46 Baum set 6 Typ "Datei"
47
48 Baum insert 3 end 7
49 Baum set 7 Name "Datei 4"
50 Baum set 7 Typ "Datei"
51
52 puts "Vorher:"
53 puts "Baum children -all root: [Baum children -all root]"
54
55 set Kinder [Baum children root]
56 foreach Kind $Kinder {
57   Baum delete $Kind
58 }
59
60 puts "Nachher:"
61 puts "Baum children -all root: [Baum children -all root]"
```

```
oliver@debian:~$ ./Beispiel549.tcl
root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)
      |               |               +- Datei 4(7)
      |               +- Datei 1(4)
      |               +- Datei 2(5)
      +- Ordner C(2)
Der Index ist in Klammern gesetzt.

Vorher:
Baum children -all root: 0 1 2 3 4 5 6 7
Nachher:
Baum children -all root:
oliver@debian:~$
```

In den Zeilen 55 bis 58 werden alle Einträge des Trees gelöscht. In Zeile 55 werden alle Elemente des Wurzelknotens `root` ermittelt. In Zeile 57 werden dann alle gefundenen Elemente gelöscht. Wenn ein Element seinerseits Kinder-Elemente hat, werden diese automatisch mit gelöscht.

Listing 17.3: Daten eines Elements anzeigen (Beispiel546.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require struct::tree
4
5  ::struct::tree Baum
6
7  # Es wird folgender Baum erfasst:
8  puts "root -+- Ordner A(0)"
9  puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
10 puts "      |               |               +- Datei 4(7)"
11 puts "      |               +- Datei 1(4)"
12 puts "      |               +- Datei 2(5)"
13 puts "      +- Ordner C(2)"
14 puts "Der Index ist in Klammern gesetzt."
15 puts ""
16
17 # Unterhalb von root:
18 Baum insert root end 0
19 Baum set 0 Name "Ordner A"
20 Baum set 0 Typ "Ordner"
21
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
24 Baum set 1 Typ "Ordner"
25
26 Baum insert root end 2
27 Baum set 2 Name "Ordner C"
```

```
28 Baum set 2 Typ "Ordner"
29
30 # Unterhalb von Ordner B:
31 Baum insert 1 end 3
32 Baum set 3 Name "Ordner D"
33 Baum set 3 Typ "Ordner"
34
35 Baum insert 1 end 4
36 Baum set 4 Name "Datei 1"
37 Baum set 4 Typ "Datei"
38
39 Baum insert 1 end 5
40 Baum set 5 Name "Datei 2"
41 Baum set 5 Typ "Datei"
42
43 # Unterhalb von Ordner D:
44 Baum insert 3 end 6
45 Baum set 6 Name "Datei 3"
46 Baum set 6 Typ "Datei"
47
48 Baum insert 3 end 7
49 Baum set 7 Name "Datei 4"
50 Baum set 7 Typ "Datei"
51
52 proc DatenAnzeigen {LokalerBaum Id} {
53   puts "ID:$Id"
54   puts "Schluessel:[$LokalerBaum keys $Id]"
55   puts "Name:[$LokalerBaum get $Id Name]"
56   puts "Typ:[$LokalerBaum get $Id Typ]"
57   puts "Tiefe unterhalb root:[$LokalerBaum depth $Id]"
58   puts "Eltern-ID:[$LokalerBaum parent $Id]"
59   puts "Kinder-ID:[$LokalerBaum children $Id]"
60   puts "Alle Kinder-ID:[$LokalerBaum children -all $Id]"
61   puts "Index innerhalb des Elternknotens:[$LokalerBaum ↩
       index $Id]"
62   puts "Naechster Knoten (innerhalb des eigenen ↩
       Elternknotens):[$LokalerBaum next $Id]"
63   puts "Vorheriger Knoten (innerhalb des eigenen ↩
       Elternknotens):[$LokalerBaum previous $Id]"
64 }
65
66 DatenAnzeigen Baum 1
67 puts "----------"
68 DatenAnzeigen Baum 5
69 puts "----------"
70 DatenAnzeigen Baum 6
```

```
oliver : bash — Konsole
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
oliver@debian:~$ ./Beispiel546.tcl
root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)
      |               |                +- Datei 4(7)
      |               +- Datei 1(4)
      |               +- Datei 2(5)
      +- Ordner C(2)
Der Index ist in Klammern gesetzt.

ID:1
Schluessel:Typ Name
Name:Ordner B
Typ:Ordner
Tiefe unterhalb root:1
Eltern-ID:root
Kinder-ID:3 4 5
Alle Kinder-ID:3 4 5 6 7
Index innerhalb des Elternknotens:1
Naechster Knoten (innerhalb des eigenen Elternknotens):2
Vorheriger Knoten (innerhalb des eigenen Elternknotens):0
----------
ID:5
Schluessel:Typ Name
Name:Datei 2
Typ:Datei
Tiefe unterhalb root:2
Eltern-ID:1
Kinder-ID:
Alle Kinder-ID:
Index innerhalb des Elternknotens:2
Naechster Knoten (innerhalb des eigenen Elternknotens):
Vorheriger Knoten (innerhalb des eigenen Elternknotens):4
----------
ID:6
Schluessel:Typ Name
Name:Datei 3
Typ:Datei
Tiefe unterhalb root:3
Eltern-ID:3
Kinder-ID:
Alle Kinder-ID:
Index innerhalb des Elternknotens:0
Naechster Knoten (innerhalb des eigenen Elternknotens):7
Vorheriger Knoten (innerhalb des eigenen Elternknotens):
oliver@debian:~$
```

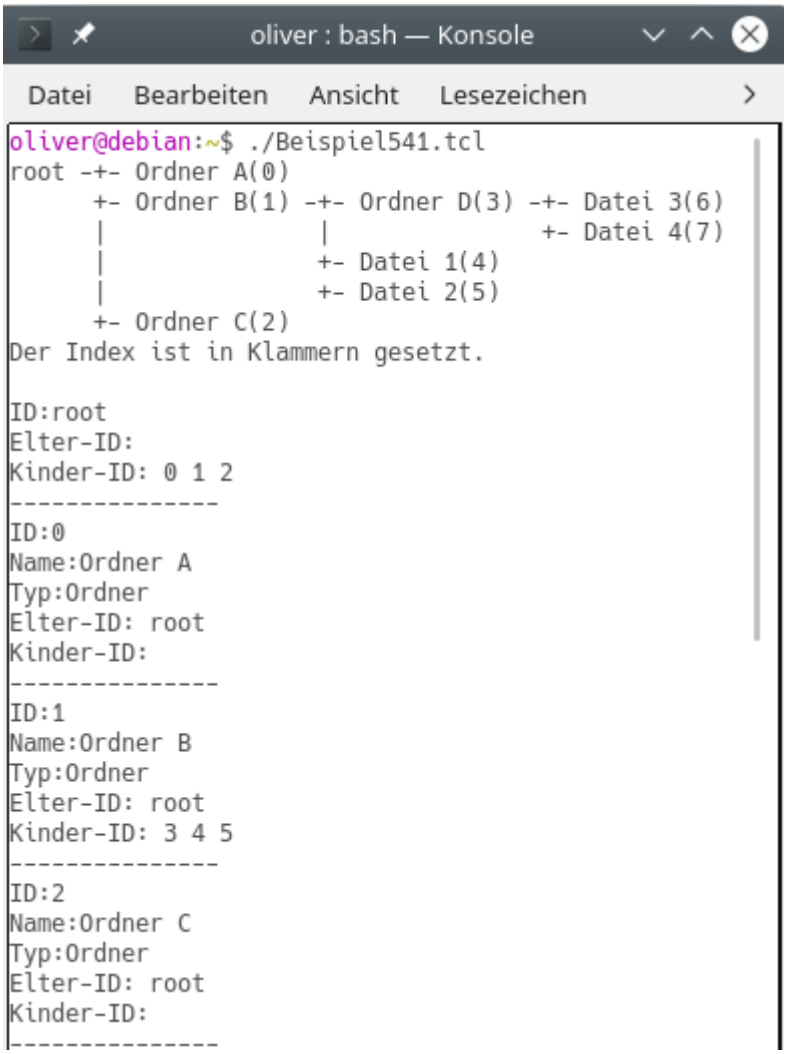
In den Zeilen 53 bis 63 werden detaillierte Daten zu einem Element angezeigt.

Listing 17.4: Elemente durchwandern (Beispiel541.tcl)

```
1 #!/usr/bin/env tclsh
```

```
2
3 package require struct::tree
4
5 ::struct::tree Baum
6
7 # Es wird folgender Baum erfasst:
8 puts "root -+- Ordner A(0)"
9 puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
10 puts "      |               |                +- Datei 4(7)"
11 puts "      |               +- Datei 1(4)"
12 puts "      |               +- Datei 2(5)"
13 puts "      +- Ordner C(2)"
14 puts "Der Index ist in Klammern gesetzt."
15 puts ""
16
17 # Unterhalb von root:
18 Baum insert root end 0
19 Baum set 0 Name "Ordner A"
20 Baum set 0 Typ "Ordner"
21
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
24 Baum set 1 Typ "Ordner"
25
26 Baum insert root end 2
27 Baum set 2 Name "Ordner C"
28 Baum set 2 Typ "Ordner"
29
30 # Unterhalb von Ordner B:
31 Baum insert 1 end 3
32 Baum set 3 Name "Ordner D"
33 Baum set 3 Typ "Ordner"
34
35 Baum insert 1 end 4
36 Baum set 4 Name "Datei 1"
37 Baum set 4 Typ "Datei"
38
39 Baum insert 1 end 5
40 Baum set 5 Name "Datei 2"
41 Baum set 5 Typ "Datei"
42
43 # Unterhalb von Ordner D:
44 Baum insert 3 end 6
45 Baum set 6 Name "Datei 3"
46 Baum set 6 Typ "Datei"
47
48 Baum insert 3 end 7
49 Baum set 7 Name "Datei 4"
50 Baum set 7 Typ "Datei"
51
52 Baum walk root -order pre -type bfs Id {
53   puts "ID:$Id"
54   if {[Baum keyexists $Id Name]} {
```

```
55      puts "Name:[Baum get $Id Name]"
56    }
57    if {[Baum keyexists $Id Typ]} {
58      puts "Typ:[Baum get $Id Typ]"
59    }
60    puts "Eltern-ID: [Baum parent $Id]"
61    puts "Kinder-ID: [Baum children $Id]"
62    puts "---------------"
63  }
```



```
ID:3
Name:Ordner D
Typ:Ordner
Elter-ID: 1
Kinder-ID: 6 7
---------------
ID:4
Name:Datei 1
Typ:Datei
Elter-ID: 1
Kinder-ID:
---------------
ID:5
Name:Datei 2
Typ:Datei
Elter-ID: 1
Kinder-ID:
---------------
ID:6
Name:Datei 3
Typ:Datei
Elter-ID: 3
Kinder-ID:
---------------
ID:7
Name:Datei 4
Typ:Datei
Elter-ID: 3
Kinder-ID:
---------------
```

In den Zeilen 52 bis 63 wird der Baum ab dem Wurzelknoten `root` durchwandert. Der Index des aktuellen Elements wird in der Variablen `Id` gespeichert und kann in den Befehlen (ab Zeile 53) verwendet werden, um weitere Daten des Elements abzufragen.

Listing 17.5: Reihenfolge des Wanderns (Beispiel542.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require struct::tree
4
5  ::struct::tree Baum
6
7  # Es wird folgender Baum erfasst:
8  puts "root -+- Ordner A(0)"
9  puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
10 puts "      |               |                +- Datei 4(7)"
11 puts "      |               +- Datei 1(4)"
12 puts "      |               +- Datei 2(5)"
13 puts "      +- Ordner C(2)"
14 puts "Der Index ist in Klammern gesetzt."
15 puts ""
16
17 # Unterhalb von root:
```

```
18 Baum insert root end 0
19 Baum set 0 Name "Ordner A"
20 Baum set 0 Typ "Ordner"
21
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
24 Baum set 1 Typ "Ordner"
25
26 Baum insert root end 2
27 Baum set 2 Name "Ordner C"
28 Baum set 2 Typ "Ordner"
29
30 # Unterhalb von Ordner B:
31 Baum insert 1 end 3
32 Baum set 3 Name "Ordner D"
33 Baum set 3 Typ "Ordner"
34
35 Baum insert 1 end 4
36 Baum set 4 Name "Datei 1"
37 Baum set 4 Typ "Datei"
38
39 Baum insert 1 end 5
40 Baum set 5 Name "Datei 2"
41 Baum set 5 Typ "Datei"
42
43 # Unterhalb von Ordner D:
44 Baum insert 3 end 6
45 Baum set 6 Name "Datei 3"
46 Baum set 6 Typ "Datei"
47
48 Baum insert 3 end 7
49 Baum set 7 Name "Datei 4"
50 Baum set 7 Typ "Datei"
51
52 puts "Optionen: -order pre -type dfs"
53 Baum walk root -order pre -type dfs Id {
54         set Text "ID:$Id"
55         if {[Baum keyexists $Id Name]} {
56                 set Text [string cat $Text " Name:[Baum ↩
                      get $Id Name]"]
57         }
58         puts $Text
59 }
60 puts "-----------------"
61 puts "Optionen: -order pre -type bfs"
62 Baum walk root -order pre -type bfs Id {
63         set Text "ID:$Id"
64         if {[Baum keyexists $Id Name]} {
65                 set Text [string cat $Text " Name:[Baum ↩
                      get $Id Name]"]
66         }
67         puts $Text
68 }
```

```
69 puts "-----------------"
70 puts "Optionen: -order post -type dfs"
71 Baum walk root -order post -type dfs Id {
72   set Text "ID:$Id"
73   if {[Baum keyexists $Id Name]} {
74     set Text [string cat $Text " Name:[Baum get $Id Name]"]
75   }
76   puts $Text
77 }
78 puts "-----------------"
79 puts "Optionen: -order post -type bfs"
80 Baum walk root -order post -type bfs Id {
81   set Text "ID:$Id"
82   if {[Baum keyexists $Id Name]} {
83     set Text [string cat $Text " Name:[Baum get $Id Name]"]
84   }
85   puts $Text
86 }
```

```
oliver : bash — Konsole
Datei   Bearbeiten   Ansicht   Lesezeichen

oliver@debian:~$ ./Beispiel542.tcl
root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)
      |               |                +- Datei 4(7)
      |               +- Datei 1(4)
      |               +- Datei 2(5)
      +- Ordner C(2)
Der Index ist in Klammern gesetzt.

Optionen: -order pre -type dfs
ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:3 Name:Ordner D
ID:6 Name:Datei 3
ID:7 Name:Datei 4
ID:4 Name:Datei 1
ID:5 Name:Datei 2
ID:2 Name:Ordner C
-----------------
Optionen: -order pre -type bfs
ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:2 Name:Ordner C
ID:3 Name:Ordner D
ID:4 Name:Datei 1
ID:5 Name:Datei 2
ID:6 Name:Datei 3
ID:7 Name:Datei 4
-----------------
```

```
-----------------
Optionen: -order post -type dfs
ID:0 Name:Ordner A
ID:6 Name:Datei 3
ID:7 Name:Datei 4
ID:3 Name:Ordner D
ID:4 Name:Datei 1
ID:5 Name:Datei 2
ID:1 Name:Ordner B
ID:2 Name:Ordner C
ID:root
-----------------
Optionen: -order post -type bfs
ID:7 Name:Datei 4
ID:6 Name:Datei 3
ID:5 Name:Datei 2
ID:4 Name:Datei 1
ID:3 Name:Ordner D
ID:2 Name:Ordner C
ID:1 Name:Ordner B
ID:0 Name:Ordner A
ID:root
oliver@debian:~$
```

In den Zeilen 52 bis 86 werden die verschiedenen Parameter für die Optionen `-order` und `-type` vorgestellt. Bei der Option `-order` gibt es die Möglichkeiten `pre` und `post`. Bei `pre` startet das Wandern beim Wurzelknoten `root`. Bei `post` wird am Ende des Baums gestartet. Bei der Option `-type` gibt es die Parameter `bfs` und `dfs`. Bei `bfs` werden zuerst alle Elemente des aktuellen Knotens durchlaufen, bevor in einen Unterknoten verzweigt wird. Bei `dfs` wird erst der Unterknoten durchlaufen, bevor das nächste Element des aktuellen Knotens aufgerufen wird.

Listing 17.6: Element verschieben (Beispiel543.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require struct::tree
4
5  ::struct::tree Baum
6
7  # Es wird folgender Baum erfasst:
8  puts "root -+- Ordner A(0)"
9  puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
10 puts "      |               |                +- Datei 4(7)"
11 puts "      |               +- Datei 1(4)"
12 puts "      |               +- Datei 2(5)"
13 puts "      +- Ordner C(2)"
14 puts "Der Index ist in Klammern gesetzt."
15 puts ""
16
17 # Unterhalb von root:
18 Baum insert root end 0
19 Baum set 0 Name "Ordner A"
20 Baum set 0 Typ "Ordner"
```

```
21
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
24 Baum set 1 Typ "Ordner"
25
26 Baum insert root end 2
27 Baum set 2 Name "Ordner C"
28 Baum set 2 Typ "Ordner"
29
30 # Unterhalb von Ordner B:
31 Baum insert 1 end 3
32 Baum set 3 Name "Ordner D"
33 Baum set 3 Typ "Ordner"
34
35 Baum insert 1 end 4
36 Baum set 4 Name "Datei 1"
37 Baum set 4 Typ "Datei"
38
39 Baum insert 1 end 5
40 Baum set 5 Name "Datei 2"
41 Baum set 5 Typ "Datei"
42
43 # Unterhalb von Ordner D:
44 Baum insert 3 end 6
45 Baum set 6 Name "Datei 3"
46 Baum set 6 Typ "Datei"
47
48 Baum insert 3 end 7
49 Baum set 7 Name "Datei 4"
50 Baum set 7 Typ "Datei"
51
52 Baum walk root -order pre -type dfs Id {
53         set Text "ID:$Id"
54         if {[Baum keyexists $Id Name]} {
55                 set Text [string cat $Text " Name:[Baum ⤶
                      get $Id Name]"]
56         }
57         puts $Text
58 }
59
60 puts ""
61 #Baum move 1 end 6
62 Baum move 1 2 6 ; # Elterknoten=1, Index innerhalb des ⤶
      neuen Elternknotens=2
63 puts "nach der Verschiebung von Datei 3 in"
64 puts "den Ordner B an Indexposition 2:"
65 puts ""
66
67 puts "root -+- Ordner A(0)"
68 puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 4(7)"
69 puts "      |               +- Datei 1(4)"
70 puts "      |               +- Datei 3(6)"
71 puts "      |               +- Datei 2(5)"
```

```
72 puts "       +- Ordner C(2)"
73 puts ""
74
75 Baum walk root -order pre -type dfs Id {
76   set Text "ID:$Id"
77   if {[Baum keyexists $Id Name]} {
78     set Text [string cat $Text " Name:[Baum get $Id Name]"]
79   }
80   puts $Text
81 }
```

```
oliver : bash — Konsole
Datei   Bearbeiten   Ansicht   Lesezeichen
oliver@debian:~$ ./Beispiel543.tcl
root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)
      |               |                +- Datei 4(7)
      |               +- Datei 1(4)
      |               +- Datei 2(5)
      +- Ordner C(2)
Der Index ist in Klammern gesetzt.

ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:3 Name:Ordner D
ID:6 Name:Datei 3
ID:7 Name:Datei 4
ID:4 Name:Datei 1
ID:5 Name:Datei 2
ID:2 Name:Ordner C

nach der Verschiebung von Datei 3 in
den Ordner B an Indexposition 2:

root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 4(7)
      |               +- Datei 1(4)
      |               +- Datei 3(6)
      |               +- Datei 2(5)
      +- Ordner C(2)

ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:3 Name:Ordner D
ID:7 Name:Datei 4
ID:4 Name:Datei 1
ID:6 Name:Datei 3
ID:5 Name:Datei 2
ID:2 Name:Ordner C
oliver@debian:~$
```

In Zeile 62 wird das Element mit dem Index 6 (Datei 3) in den Elternknoten mit dem Index 1 (Ordner B) verschoben. Das Element wird innerhalb diese Elternknotens an der Indexposition 2 eingefügt.

Listing 17.7: Element löschen (Beispiel544.tcl)

```
1 #!/usr/bin/env tclsh
2
3 package require struct::tree
4
```

```
5  ::struct::tree Baum
6
7  # Es wird folgender Baum erfasst:
8  puts "root -+- Ordner A(0)"
9  puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
10 puts "      |               |                +- Datei 4(7)"
11 puts "      |               +- Datei 1(4)"
12 puts "      |               +- Datei 2(5)"
13 puts "      +- Ordner C(2)"
14 puts "Der Index ist in Klammern gesetzt."
15 puts ""
16
17 # Unterhalb von root:
18 Baum insert root end 0
19 Baum set 0 Name "Ordner A"
20 Baum set 0 Typ "Ordner"
21
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
24 Baum set 1 Typ "Ordner"
25
26 Baum insert root end 2
27 Baum set 2 Name "Ordner C"
28 Baum set 2 Typ "Ordner"
29
30 # Unterhalb von Ordner B:
31 Baum insert 1 end 3
32 Baum set 3 Name "Ordner D"
33 Baum set 3 Typ "Ordner"
34
35 Baum insert 1 end 4
36 Baum set 4 Name "Datei 1"
37 Baum set 4 Typ "Datei"
38
39 Baum insert 1 end 5
40 Baum set 5 Name "Datei 2"
41 Baum set 5 Typ "Datei"
42
43 # Unterhalb von Ordner D:
44 Baum insert 3 end 6
45 Baum set 6 Name "Datei 3"
46 Baum set 6 Typ "Datei"
47
48 Baum insert 3 end 7
49 Baum set 7 Name "Datei 4"
50 Baum set 7 Typ "Datei"
51
52 Baum walk root -order pre -type dfs Id {
53         set Text "ID:$Id"
54         if {[Baum keyexists $Id Name]} {
55                 set Text [string cat $Text " Name:[Baum ↩
                      get $Id Name]"]
56         }
```

```
57          puts $Text
58 }
59
60 puts ""
61 Baum delete 4
62 puts "nach dem Loeschen von Datei 1:"
63 puts ""
64
65 puts "root -+- Ordner A(0)"
66 puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
67 puts "      |               |               +- Datei 4(7)"
68 puts "      |               +- Datei 2(5)"
69 puts "      +- Ordner C(2)"
70
71 Baum walk root -order pre -type dfs Id {
72   set Text "ID:$Id"
73   if {[Baum keyexists $Id Name]} {
74     set Text [string cat $Text " Name:[Baum get $Id Name]"
       ]
75   }
76   puts $Text
77 }
78
79 puts ""
80 Baum delete 1
81 puts "nach dem Loeschen von Ordner B:"
82 puts ""
83
84 puts "root -+- Ordner A(0)"
85 puts "      +- Ordner C(2)"
86
87 Baum walk root -order pre -type dfs Id {
88   set Text "ID:$Id"
89   if {[Baum keyexists $Id Name]} {
90     set Text [string cat $Text " Name:[Baum get $Id Name]"
       ]
91   }
92   puts $Text
93 }
```

```
oliver@debian:~$ ./Beispiel544.tcl
root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)
      |               |               +- Datei 4(7)
      |               +- Datei 1(4)
      |               +- Datei 2(5)
      +- Ordner C(2)
Der Index ist in Klammern gesetzt.

ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:3 Name:Ordner D
ID:6 Name:Datei 3
ID:7 Name:Datei 4
ID:4 Name:Datei 1
ID:5 Name:Datei 2
ID:2 Name:Ordner C

nach dem Löschen von Datei 1:

root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)
      |               |               +- Datei 4(7)
      |               +- Datei 2(5)
      +- Ordner C(2)
ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:3 Name:Ordner D
ID:6 Name:Datei 3
ID:7 Name:Datei 4
ID:5 Name:Datei 2
ID:2 Name:Ordner C

nach dem Löschen von Ordner B:

root -+- Ordner A(0)
      +- Ordner C(2)
ID:root
ID:0 Name:Ordner A
ID:2 Name:Ordner C
oliver@debian:~$
```

In Zeile 61 wird das Element mit dem Index 4 (Datei 1) gelöscht. In Zeile 80 wird das Element mit dem Index 1 (Ordner B) gelöscht. Dadurch werden auch alle Elemente unterhalb des Ordners B gelöscht.

Listing 17.8: Element ausschneiden (Beispiel545.tcl)

```
1 #!/usr/bin/env tclsh
```

```
2
3 package require struct::tree
4
5 ::struct::tree Baum
6
7 # Es wird folgender Baum erfasst:
8 puts "root -+- Ordner A(0)"
9 puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
10 puts "      |               |                +- Datei 4(7)"
11 puts "      |               +- Datei 1(4)"
12 puts "      |               +- Datei 2(5)"
13 puts "      +- Ordner C(2)"
14 puts "Der Index ist in Klammern gesetzt."
15 puts ""
16
17 # Unterhalb von root:
18 Baum insert root end 0
19 Baum set 0 Name "Ordner A"
20 Baum set 0 Typ "Ordner"
21
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
24 Baum set 1 Typ "Ordner"
25
26 Baum insert root end 2
27 Baum set 2 Name "Ordner C"
28 Baum set 2 Typ "Ordner"
29
30 # Unterhalb von Ordner B:
31 Baum insert 1 end 3
32 Baum set 3 Name "Ordner D"
33 Baum set 3 Typ "Ordner"
34
35 Baum insert 1 end 4
36 Baum set 4 Name "Datei 1"
37 Baum set 4 Typ "Datei"
38
39 Baum insert 1 end 5
40 Baum set 5 Name "Datei 2"
41 Baum set 5 Typ "Datei"
42
43 # Unterhalb von Ordner D:
44 Baum insert 3 end 6
45 Baum set 6 Name "Datei 3"
46 Baum set 6 Typ "Datei"
47
48 Baum insert 3 end 7
49 Baum set 7 Name "Datei 4"
50 Baum set 7 Typ "Datei"
51
52 Baum walk root -order pre -type dfs Id {
53   set Text "ID:$Id"
54   if {[Baum keyexists $Id Name]} {
```

```
55      set Text [string cat $Text " Name:[Baum get $Id Name]"]
56    }
57    puts $Text
58  }
59
60  puts ""
61  Baum cut 3
62  puts "nach dem Ausschneiden von Ordner D:"
63  puts ""
64
65  puts "root -+- Ordner A(0)"
66  puts "      +- Ordner B(1) -+- Datei 3(6)"
67  puts "      |              -+- Datei 4(7)"
68  puts "      |               +- Datei 1(4)"
69  puts "      |               +- Datei 2(5)"
70  puts "      +- Ordner C(2)"
71  puts ""
72
73  Baum walk root -order pre -type dfs Id {
74    set Text "ID:$Id"
75    if {[Baum keyexists $Id Name]} {
76      set Text [string cat $Text " Name:[Baum get $Id Name]"]
77    }
78    puts $Text
79  }
```

```
oliver : bash — Konsole
Datei   Bearbeiten   Ansicht   Lesezeichen

oliver@debian:~$ ./Beispiel545.tcl
root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)
      |               |                +- Datei 4(7)
      |               +- Datei 1(4)
      |               +- Datei 2(5)
      +- Ordner C(2)
Der Index ist in Klammern gesetzt.

ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:3 Name:Ordner D
ID:6 Name:Datei 3
ID:7 Name:Datei 4
ID:4 Name:Datei 1
ID:5 Name:Datei 2
ID:2 Name:Ordner C

nach dem Ausschneiden von Ordner D:

root -+- Ordner A(0)
      +- Ordner B(1) -+- Datei 3(6)
      |              -+- Datei 4(7)
      |               +- Datei 1(4)
      |               +- Datei 2(5)
      +- Ordner C(2)

ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:6 Name:Datei 3
ID:7 Name:Datei 4
ID:4 Name:Datei 1
ID:5 Name:Datei 2
ID:2 Name:Ordner C
oliver@debian:~$
```

In Zeile 61 wird das Element mit dem Index 3 (Ordner D) ausgeschnitten. Dadurch werden alle Elemente unterhalb des Ordners D an dessen Elternknoten (Ordner B) verschoben.

Listing 17.9: Elemente tauschen (Beispiel547.tcl)

```
1 #!/usr/bin/env tclsh
2
3 package require struct::tree
4
5 ::struct::tree Baum
6
7 # Es wird folgender Baum erfasst:
8 puts "root -+- Ordner A(0)"
```

```
9  puts "       +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)"
10 puts "       |               |                +- Datei 4(7)"
11 puts "       |               +- Datei 1(4)"
12 puts "       |               +- Datei 2(5)"
13 puts "       +- Ordner C(2)"
14 puts "Der Index ist in Klammern gesetzt."
15 puts ""
16
17 # Unterhalb von root:
18 Baum insert root end 0
19 Baum set 0 Name "Ordner A"
20 Baum set 0 Typ "Ordner"
21
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
24 Baum set 1 Typ "Ordner"
25
26 Baum insert root end 2
27 Baum set 2 Name "Ordner C"
28 Baum set 2 Typ "Ordner"
29
30 # Unterhalb von Ordner B:
31 Baum insert 1 end 3
32 Baum set 3 Name "Ordner D"
33 Baum set 3 Typ "Ordner"
34
35 Baum insert 1 end 4
36 Baum set 4 Name "Datei 1"
37 Baum set 4 Typ "Datei"
38
39 Baum insert 1 end 5
40 Baum set 5 Name "Datei 2"
41 Baum set 5 Typ "Datei"
42
43 # Unterhalb von Ordner D:
44 Baum insert 3 end 6
45 Baum set 6 Name "Datei 3"
46 Baum set 6 Typ "Datei"
47
48 Baum insert 3 end 7
49 Baum set 7 Name "Datei 4"
50 Baum set 7 Typ "Datei"
51
52 Baum walk root -order pre -type dfs Id {
53   set Text "ID:$Id"
54   if {[Baum keyexists $Id Name]} {
55     set Text [string cat $Text " Name:[Baum get $Id Name]"]
56   }
57   puts $Text
58 }
59
60 puts ""
```

```
Baum swap 4 6
puts "nach dem Vertauschen von Datei 1 mit Datei 3:"
puts ""

puts "root -+- Ordner A(0)"
puts "      +- Ordner B(1) -+- Ordner D(3) -+- Datei 1(4)"
puts "      |               |               +- Datei 4(7)"
puts "      |               +- Datei 3(6)"
puts "      |               +- Datei 2(5)"
puts "      +- Ordner C(2)"

Baum walk root -order pre -type dfs Id {
  set Text "ID:$Id"
  if {[Baum keyexists $Id Name]} {
    set Text [string cat $Text " Name:[Baum get $Id Name]"]
  }
  puts $Text
}

puts ""
Baum swap 2 3
puts "nach dem Vertauschen von Ordner C mit Ordner D:"
puts ""

puts "root -+- Ordner A(0)"
puts "      +- Ordner B(1) -+- Ordner C(2) -+- Datei 1(4)"
puts "      |               |               +- Datei 4(7)"
puts "      |               +- Datei 3(6)"
puts "      |               +- Datei 2(5)"
puts "      +- Ordner D(3)"

Baum walk root -order pre -type dfs Id {
  set Text "ID:$Id"
  if {[Baum keyexists $Id Name]} {
    set Text [string cat $Text " Name:[Baum get $Id Name]"]
  }
  puts $Text
}
```

oliver : bash — Konsole

Datei Bearbeiten Ansicht Lesezeichen

```
oliver@debian:~$ ./Beispiel547.tcl
root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 3(6)
      |               |               +- Datei 4(7)
      |               +- Datei 1(4)
      |               +- Datei 2(5)
      +- Ordner C(2)
Der Index ist in Klammern gesetzt.

ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:3 Name:Ordner D
ID:6 Name:Datei 3
ID:7 Name:Datei 4
ID:4 Name:Datei 1
ID:5 Name:Datei 2
ID:2 Name:Ordner C

nach dem Vertauschen von Datei 1 mit Datei 3:

root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner D(3) -+- Datei 1(4)
      |               |               +- Datei 4(7)
      |               +- Datei 3(6)
      |               +- Datei 2(5)
      +- Ordner C(2)
ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:3 Name:Ordner D
ID:4 Name:Datei 1
ID:7 Name:Datei 4
ID:6 Name:Datei 3
ID:5 Name:Datei 2
ID:2 Name:Ordner C

nach dem Vertauschen von Ordner C mit Ordner D:

root -+- Ordner A(0)
      +- Ordner B(1) -+- Ordner C(2) -+- Datei 1(4)
      |               |               +- Datei 4(7)
      |               +- Datei 3(6)
      |               +- Datei 2(5)
      +- Ordner D(3)
ID:root
ID:0 Name:Ordner A
ID:1 Name:Ordner B
ID:2 Name:Ordner C
ID:4 Name:Datei 1
ID:7 Name:Datei 4
ID:6 Name:Datei 3
ID:5 Name:Datei 2
ID:3 Name:Ordner D
oliver@debian:~$
```

In Zeile 61 werden die Elemente mit dem Index 4 (Datei 1) und Index 6 (Datei 3) vertauscht. In Zeile 81 werden die Elemente mit dem Index 2 (Ordner C) und Index 3 (Ordner D) vertauscht. Die Unterelemente der Ordner werden dabei nicht mitgenommen.

17.1 Tree mit Treeview kombinieren

In Kapitel 41.17.1 wird beschrieben, wie Sie einen `Tree` mit einem `Treeview` anzeigen können.

18 Graph (Gerichteter Graph)

Mit Hilfe eines gerichtetem Graphen können zum Beispiel Prozess- oder Projektabläufe abgebildet werden. Ein gerichteter Graph besteht aus einer Menge von Knoten und einer Menge von Kanten. Eine Kante verbindet zwei Knoten miteinander und kann immer nur in eine Richtung durchlaufen werden (von Knoten 1 zu Knoten2). Es ist aber zulässig zwei Knoten wechselseitig mit zwei Kanten, die gegensätzlich gerichtet sind, zu verbinden. Man darf auch einen Knoten über eine Kante mit sich selbst verbinden.

Je nachdem wie man den Graphen erzeugt, werden die Graph-Befehle verwendet. In der ersten Variante wird der Graph mit dem Befehl `::struct::graph MeinGraph` erzeugt. In Folge dessen wird ein globales Objekt namens `MeinGraph` erstellt, so dass alle Graph-Befehle mit `MeinGraph` beginnen (z. B. `MeinGraph destroy`). In der zweiten Variante wird der Graph mit dem Befehl `set MeinGraph [::struct::graph]` erzeugt. Dadurch wird das Graph-Objekt in der Variablen `MeinGraph` gespeichert. Die Graph-Befehle beginnen deshalb mit `$MeinGraph` (z. B. `$MeinGraph destroy`).

Tabelle 18.1: Die wichtigsten Befehle für den Pool (gemäß Variante 1)

Befehl	Beschreibung
`package require struct::graph`	Bindet das Graph-Paket in das Programm ein
`::struct::graph MeinGraph`	Erzeugt einen leeren Graphen (Variante 1)
`MeinGraph node insert Knoten1 Knoten2`	Fügt dem Graphen Knoten hinzu
`MeinGraph node set Knoten Schlüssel Wert`	Fügt dem Knoten ein Schlüssel-Wert-Paar hinzu
`MeinGraph node get Knoten Schlüssel`	Holt den Wert zum Schlüssel des Knotens
`MeinGraph node degree Knoten`	Anzahl der Kanten am Knoten
`MeinGraph node degree -in Knoten`	Anzahl der eingehenden Kanten am Knoten
`MeinGraph node degree -out Knoten`	Anzahl der abgehenden Kanten am Knoten
`MeinGraph node delete Knoten`	Löscht den Knoten mit allen seinen Kanten
`MeinGraph node exists Knoten`	Prüft, ob der Knoten existiert

Tabelle 18.1: Die wichtigsten Befehle für den Pool (gemäß Variante 1)

Befehl	Beschreibung
`MeinGraph node keys Knoten`	Ermittelt alle Schlüssel des Knotens
`MeinGraph node keyexists Knoten`	Prüft, ob der Schlüssel des Knotens existiert
`MeinGraph node opposite Knoten Kante`	Ermittelt den Knoten, der am anderen Ende der Kante des Knotens ist
`MeinGraph node rename Knote NeuerName`	Benennt den Knoten um.
`MeinGraph node unset Knoten Schlüssel`	Entfernt den Schlüssel von dem Knoten
`MeinGraph nodes`	Erstellt eine Liste aller Knoten. Es gibt zu diesem Befehl viele Optionen (siehe dazu die Internetseite)
`MeinGraph swap Knoten1 Knoten2`	Vertauscht Knoten1 mit Knoten2
`MeinGraph arc insert Knoten1 Knoten2`	Erstellt eine gerichtete Kante von Knoten1 zu Knoten2
`MeinGraph arc set Kante Schlüssel Wert`	Fügt der Kante ein Schlüssel-Wert-Paar hinzu
`MeinGraph arc source Kante`	Ermittelt den Knoten, bei dem die Kante beginnt
`MeinGraph arc target Kante`	Ermittelt den Knoten, bei dem die Kante endet
`MeinGraph arc nodes Kante`	Ermittelt die beiden Knoten, die die Kante verbindet (Rückgabe ist eine Liste aus zwei Elementen)
`MeinGraph arc move-source Kante Startknoten`	Ändert den Startknoten der Kante in einen anderen Knoten
`MeinGraph arc move-target Kante Zielknoten`	Ändert den Zielknoten der Kante in einen anderen Knoten
`MeinGraph arc move Kante Startknoten Zielknoten`	Ändert Start- und Zielknoten der Kante
`MeinGraph arc unset Kante Schlüssel`	Entfernt den Schlüssel von der Kante

Tabelle 18.1: Die wichtigsten Befehle für den Pool (gemäß Variante 1)

Befehl	Beschreibung
`MeinGraph arcs`	Erstellt eine Liste aller Kanten. Es gibt zu diesem Befehl viele Optionen (siehe dazu die Internetseite)

`MeinGraph walk Startknoten`
`-type Typ`
`-order Reihenfolge`
`-dir forward -command Befehl`
	Durchläuft den Graphen ab dem Startknoten und führt für jeden Knoten den Befehl aus. Es gibt folgende Optionen (Erklärung siehe nächste Tabelle)
	`-type bfs`
	`-type dfs`
	`-order pre`
	`-order post`
	`-order both`
	`-dir backward`
	`-dir forward`

`set Var [MeinGraph serialize]`	Serialisiert den Graphen, so dass man den Graphen zum Beispiel in eine Datei speichern kann
`MeinGraph deserialize $Var`	Wenn man den Graphen zuvor serialisiert hat, kann man den Graphen mit diesem Befehl wieder herstellen
`MeinGraph destroy`	Löscht den Graphen

Es gibt noch weitere Optionen. Siehe dazu
`https://tools.ietf.org/doc/tcllib/html/graph.html`

Tabelle 18.2: Die Optionen beim walk-Befehl

Option	Beschreibung
`-type bfs`	Die Art des Durchlaufs. `bfs` (breadth-first) bedeutet, zuerst in die Breite, dann in die Tiefe
`-type dfs`	Die Art des Durchlaufs. `dfs` (depth-first) bedeutet, zuerst in die Tiefe, dann in die Breite (default-Wert)
`-order pre`	Die Reihenfolge des Laufs. `pre` bedeutet, dass ein Knoten vor allen seinen Nachbarn besucht wird (default-Wert).
`-order post`	Die Reihenfolge des Laufs. `post` bedeutet, dass ein Elternknoten nach allen seinen Nachbarn besucht wird. Die Kombination mit `-type bfs` ist unzulässig.

Tabelle 18.2: Die Optionen beim walk-Befehl

Option	Beschreibung
`-order both`	Die Reihenfolge des Laufs. `both` bedeutet, dass ein Knoten vor und nach einem seiner Nachbarn besucht wird. Die Kombination mit `-type bfs` ist unzulässig.
`-dir forward`	Die Richtung der Wanderung. `forward` bedeutet, in Richtung der abgehenden Kanten
`-dir backward`	Die Richtung der Wanderung. `backward` bedeutet, entgegen der Richtung der ankommenden Kanten

Listing 18.1: Graph (Beispiel562.tcl)

```
#!/usr/bin/env tclsh

package require struct::graph

proc KnotenAnzeigen {Modus Graph Knoten} {
  # Zeigt die Daten des Knoten an
  global Zeitformat
  global AngezeigteKnoten

  if {$Knoten in $AngezeigteKnoten} {return}

  puts "Knoten=$Knoten"
  puts "  Beschreibung:[$Graph node get $Knoten Beschreibung]"
  puts "  Person:[$Graph node get $Knoten Person]"
  set Beginn [$Graph node get $Knoten Beginn]
  puts "  Beginn:[clock format $Beginn -format $Zeitformat]"
  puts "  Dauer:[$Graph node get $Knoten Dauer] Woche(n)"
  set Ende [$Graph node get $Knoten Ende]
  puts "  Ende=[clock format $Ende -format $Zeitformat]"

  puts "vorherige(r) Knoten:"
  foreach Kante [$Graph arcs -in $Knoten] {
    puts "  [$Graph arc source $Kante]"
  }

  puts "folgende(r) Knoten:"
  foreach Kante [$Graph arcs -out $Knoten] {
    puts "  [$Graph arc target $Kante]"
  }

  lappend AngezeigteKnoten $Knoten
  puts "-----------------"
}
```

```
34
35 proc TermineBerechnen {Modus Graph Knoten} {
36   # Berechnet Beginn und Ende des Knoten
37   global BerechneteKnoten
38
39   if {$Knoten ni $BerechneteKnoten} {
40     # Das maximale Ende aus allen vorherigen Knoten ↩
       bestimmen
41     set EndeMax 0
42     foreach Kante [$Graph arcs -in $Knoten] {
43       set KnotenVorher [$Graph arc source $Kante]
44       set Ende [$Graph node get $KnotenVorher Ende]
45       if {$Ende > $EndeMax} {
46         set EndeMax $Ende
47       }
48     }
49     if {$EndeMax != 0} {
50       $Graph node set $Knoten Beginn $EndeMax
51     }
52     set Beginn [$Graph node get $Knoten Beginn]
53     set Dauer [$Graph node get $Knoten Dauer]
54     set Ende [clock add $Beginn $Dauer weeks]
55     $Graph node set $Knoten Ende $Ende
56     lappend BerechneteKnoten $Knoten
57   }
58 }
59
60 proc Speichern {Graph Datei} {
61   # Speichert den Graph in die Datei
62   # Graph = Name des Graph
63   # Datei = Dateiname
64   set f [open $Datei w]
65   puts $f [$Graph serialize]
66   close $f
67 }
68
69 proc Oeffnen {Graph Datei} {
70   # Oeffnet die Datei und speichert die Daten als Graph
71   # Graph = Name des Graph
72   # Datei = Dateiname
73   if {[file exist $Datei] == 0} {return}
74   set f [open $Datei r]
75   gets $f Zeile
76   $Graph deserialize $Zeile
77   close $f
78 }
79
80 set Zeitformat "%d.%m.%Y"
81
82 ::struct::graph Hausbau
83
84 # Knoten erzeugen
85 Hausbau node insert Start Bodenplatte Rohbau Dach Estrich ↩
```

```
Fenster Heizung Elektrik Bodenbelag Tapete Sanitaer Einzug Ende

# Eigenschaften der Knoten festlegen
Hausbau node set Start Beschreibung "Start"
Hausbau node set Start Dauer 0 ; # in Wochen
Hausbau node set Start Person ""
Hausbau node set Start Status ""
Hausbau node set Start Beginn [clock scan "00:00:00 2021-05-01"]
Hausbau node set Start Ende ""

Hausbau node set Bodenplatte Beschreibung "Bodenplatte giessen"
Hausbau node set Bodenplatte Dauer 2
Hausbau node set Bodenplatte Person "Anton"
Hausbau node set Bodenplatte Status "fertig"
Hausbau node set Bodenplatte Beginn ""
Hausbau node set Bodenplatte Ende ""

Hausbau node set Rohbau Beschreibung "Rohbau erstellen"
Hausbau node set Rohbau Dauer 8
Hausbau node set Rohbau Person "Berta"
Hausbau node set Rohbau Status "fertig"
Hausbau node set Rohbau Beginn ""
Hausbau node set Rohbau Ende ""

Hausbau node set Dach Beschreibung "Dach decken"
Hausbau node set Dach Dauer 1
Hausbau node set Dach Person "Caesar"
Hausbau node set Dach Status "fertig"
Hausbau node set Dach Beginn ""
Hausbau node set Dach Ende ""

Hausbau node set Estrich Beschreibung "Estrich auftragen"
Hausbau node set Estrich Dauer 8
Hausbau node set Estrich Person "Dora"
Hausbau node set Estrich Status "fertig"
Hausbau node set Estrich Beginn ""
Hausbau node set Estrich Ende ""

Hausbau node set Fenster Beschreibung "Fenster einbauen"
Hausbau node set Fenster Dauer 2
Hausbau node set Fenster Person "Emil"
Hausbau node set Fenster Status "fertig"
Hausbau node set Fenster Beginn ""
Hausbau node set Fenster Ende ""

Hausbau node set Heizung Beschreibung "Heizung einbauen"
Hausbau node set Heizung Dauer 3
Hausbau node set Heizung Person "Friedrich"
Hausbau node set Heizung Status "fertig"
Hausbau node set Heizung Beginn ""
```

```
135 Hausbau node set Heizung Ende ""
136
137 Hausbau node set Elektrik Beschreibung "Elektrik ↩
    installieren"
138 Hausbau node set Elektrik Dauer 2
139 Hausbau node set Elektrik Person "Gustav"
140 Hausbau node set Elektrik Status "in Arbeit"
141 Hausbau node set Elektrik Beginn ""
142 Hausbau node set Elektrik Ende ""
143
144 Hausbau node set Bodenbelag Beschreibung "Bodenbelag ↩
    verlegen"
145 Hausbau node set Bodenbelag Dauer 3
146 Hausbau node set Bodenbelag Person "Heinrich"
147 Hausbau node set Bodenbelag Status "in Arbeit"
148 Hausbau node set Bodenbelag Beginn ""
149 Hausbau node set Bodenbelag Ende ""
150
151 Hausbau node set Tapete Beschreibung "Tapete anbringen"
152 Hausbau node set Tapete Dauer 1
153 Hausbau node set Tapete Person "Ida"
154 Hausbau node set Tapete Status "noch nicht begonnen"
155 Hausbau node set Tapete Beginn ""
156 Hausbau node set Tapete Ende ""
157
158 Hausbau node set Sanitaer Beschreibung "Sanitaer montieren↩
    "
159 Hausbau node set Sanitaer Dauer 1
160 Hausbau node set Sanitaer Person "Karla"
161 Hausbau node set Sanitaer Status "noch nicht begonnen"
162 Hausbau node set Sanitaer Beginn ""
163 Hausbau node set Sanitaer Ende ""
164
165 Hausbau node set Einzug Beschreibung "Einzug"
166 Hausbau node set Einzug Dauer 1
167 Hausbau node set Einzug Person "Ludwig"
168 Hausbau node set Einzug Status "noch nicht begonnen"
169 Hausbau node set Einzug Beginn ""
170 Hausbau node set Einzug Ende ""
171
172 Hausbau node set Ende Beschreibung "Ende"
173 Hausbau node set Ende Dauer 0
174 Hausbau node set Ende Person ""
175 Hausbau node set Ende Status ""
176 Hausbau node set Ende Beginn ""
177 Hausbau node set Ende Ende ""
178
179 # Kanten erzeugen
180 Hausbau arc insert Start Bodenplatte
181 Hausbau arc insert Bodenplatte Rohbau
182 Hausbau arc insert Rohbau Dach
183 Hausbau arc insert Dach Fenster
184 Hausbau arc insert Dach Heizung
```

```
185 Hausbau arc insert Fenster Estrich
186 Hausbau arc insert Heizung Estrich
187 Hausbau arc insert Estrich Elektrik
188 Hausbau arc insert Estrich Bodenbelag
189 Hausbau arc insert Estrich Tapete
190 Hausbau arc insert Estrich Sanitaer
191 Hausbau arc insert Elektrik Einzug
192 Hausbau arc insert Bodenbelag Einzug
193 Hausbau arc insert Tapete Einzug
194 Hausbau arc insert Sanitaer Einzug
195 Hausbau arc insert Einzug Ende
196
197 # Termine berechnen
198 set BerechneteKnoten {}
199 Hausbau walk Start -type bfs -order pre -dir forward ↩
    -command TermineBerechnen
200
201 # Graph anzeigen
202 set AngezeigteKnoten {}
203 Hausbau walk Start -type bfs -order pre -dir forward ↩
    -command KnotenAnzeigen
204
205 puts "### speichern und oeffnen als Hausbau2 ###"
206 # Graph in Variable speichern und in einen neuen Graph ↩
    importieren
207 Speichern Hausbau Graph.txt
208 ::struct::graph Hausbau2
209 Oeffnen Hausbau2 Graph.txt
210 puts "Start:[clock format [Hausbau2 node get Start Beginn]↩
     -format $Zeitformat]"
211 puts "Ende:[clock format [Hausbau2 node get Ende Ende] ↩
    -format $Zeitformat]"
```

```
oliver@ubuntu: ~
oliver@ubuntu:~$ ./Beispiel562.tcl
Knoten=Start
  Beschreibung:Start
  Person:
  Beginn:01.05.2021
  Dauer:0 Woche(n)
  Ende=01.05.2021
vorherige(r) Knoten:
folgende(r) Knoten:
  Bodenplatte
-----------------
Knoten=Bodenplatte
  Beschreibung:Bodenplatte giessen
  Person:Anton
  Beginn:01.05.2021
  Dauer:2 Woche(n)
  Ende=15.05.2021
vorherige(r) Knoten:
  Start
folgende(r) Knoten:
  Rohbau
-----------------
```

In Zeile 3 wird das Paket `graph` eingebunden. Die Prozedur in den Zeilen 5 bis 33 zeigt den Knoten an. Dabei werden an die Prozedur der `Modus`, `Graph` und `Knoten` übergeben. Der `Modus` gibt an, ob beim Aufruf der Prozedur der Knoten betreten oder verlassen wird. Entsprechend kennt der Modus die beiden Ausprägungen `enter` und `leave`. `Graph` ist der Name des Graphen und `Knoten` ist der aktuelle Knoten. Die Prozedur in den Zeilen 35 bis 58 berechnet für jeden Knoten den Beginn und das Ende. Die beiden Datumsangaben hängen von den vorherigen Knoten und der Dauer des aktuellen Knotens ab. Zeile 42 ermittelt alle Kanten, die in den aktuellen Knoten hineingehen. Die Kanten werden dann einzeln durchlaufen, um alle vorherigen Knoten zu finden. Zeile 43 ermittelt den vorherigen Knoten zu der aktuellen Kante. In Zeile 44 wird das Ende-Datum des vorherigen Knotens genommen. Wenn es älter ist als das bisher ällteste Ende-Datum der vorherigen Knoten, wird das Ende-Datum erhöht (Zeilen 45 bisd 47). In Zeile 50 wird das Beginn-Datum des aktuellen Knotens auf das Ende-Datum gemäß des am spätesten endenden vorherigen Knotens gesetzt. In den Zeilen 52 bis 55 wird mittels der Dauer des aktuellen Knotens das Ende-Datum berechnet. In Zeile 56 wird der aktuelle Knoten der Liste mit den bereits berechneten Knoten hinzugefügt, damit der Knoten nicht noch einmal berechnet wird. Der Grund hierfür ist, dass beim Durchlaufen des Graphen ein Knoten (je nach Kantenverbindungen) mehrfach aufgerufen werden kann. Die Prozedur in den Zeilen 60 bis 66 speichert den Graphen in einer Textdatei. Dazu wird der Graph serialisiert (Zeile 65). Die Prozedur in den Zeilen 69 bis 78 öffnet die Datei mit dem gespeicherten Graphen und deserialisiert die Daten (Zeile 76), so dass ein neuer Graph entsteht.

In Zeile 82 wird ein neuer Graph erzeugt. In Zeile 85 werden dem Graphen die Knoten hinzugefügt. In den Zeilen 87 bis 177 werden die Knoten definiert. In den Zeilen 179 bis 195 werden die Kanten definiert. In Zeile 199 wird der Graph durchlaufen, und für jeden Knoten wird das Beginn- und Ende-Datum berechnet. In Zeile 203 wid der Graph erneut durchlaufen und jeder Knoten mit seinen Daten angezeigt. In Zeile 207 wird der Graph gespeichert. In Zeile 208 wird eine neuer (leerer) Graph erstellt und in Zeile 209 werden die gespeicherten Graph-Daten in den neuen Graphen geladen. In den Zeilen 210 und 211

werden Beginn und Ende des neuen Graphen gezeigt.

19 Matrix

Befehle:

- `package require struct::matrix`
- `package require csv`
- `::struct::matrix MeineMatrix` oder `set MeineMatrix [::struct::matrix]`
- `csv::read2matrix -alternate $f MeineMatrix "," auto`
- `csv::writematrix MeineMatrix $f ";"`

Eine Matrix ist wie eine Tabelle. Sie besteht aus Zeilen und Spalten und jede Zelle enthält einen Wert. Man kann mit einer Matrix sehr einfach eine csv-Datei einlesen bzw. eine Matrix als csv-Datei speichern. Mit diversen Befehlen kann man auf die Zeilen, Spalten und Zellen der Matrix zugreifen und die Werte abfragen oder verändern. Die Zeilen bzw. Spalten starten mit dem Index 0, d. h. die erste Zeile hat den Index 0, die zweite Zeile den Index 1.

Wie beim Stack (siehe Kapitel 15 auf Seite 171) gibt es zwei Möglichkeiten, die Matrix zu erzeugen. In der ersten Variante wird die Matrix mit dem Befehl `::struct::matrix MeineMatrix` erzeugt. In Folge dessen wird ein globales Objekt namens `MeineMatrix` erstellt, so dass alle Matrix-Befehle mit `MeineMatrix` beginnen (z. B. `MeineMatrix rows`). In der zweiten Variante wird die Matrix mit dem Befehl `set MeineMatrix [::struct::matrix]` erzeugt. Dadurch wird das Matrix-Objekt in der Variablen `MeineMatrix` gespeichert. Die Matrix-Befehle beginnen deshalb mit `$MeineMatrix` (z. B. `$MeineMatrix rows`). Die folgende Tabelle zeigt die Befehle gemäß Variante1. Bei Variante 2 muss man nur ein `$`-Zeichen vor den Namen der Matrix setzen.

Tabelle 19.1: Die wichtigsten Befehle für eine Matrix (gemäß Variante 1)

Befehl	Beschreibung
`package require struct::matrix`	Bindet das Matrix-Paket in das Programm ein
`::struct::matrix MeineMatrix`	Erzeugt eine leere Matrix (Variante 1)
`set MeineMatrix [::struct::matrix]`	Erzeugt eine leere Matrix (Variante 2)
`MeineMatrix rows`	Ermittelt die Anzahl der Zeilen
`MeineMatrix columns`	Ermittelt die Anzahl der Spalten
`MeineMatrix cells`	Ermittelt die Anzahl der Zellen
`MeineMatrix add rows Anzahl`	Fügt leere Zeilen hinzu

Tabelle 19.1: Die wichtigsten Befehle für eine Matrix (gemäß Variante 1)

Befehl	Beschreibung
`MeineMatrix add columns Anzahl`	Fügt leere Spalten hinzu
`MeineMatrix add row $Liste`	Fügt eine Liste mit Spaltenwerten als neue Zeile hinzu
`MeineMatrix add column $Liste`	Fügt eine Liste mit Zeilenwerten als neue Spalte hinzu
`MeineMatrix insert row Zeilenindex $Liste`	Fügt eine Liste mit Spaltenwerten als neue Zeile ein
`MeineMatrix insert column Spaltenindex $Liste`	Fügt eine Liste mit Zeilenwerten als neue Spalte ein
`MeineMatrix set cell Spaltenindex Zeilenindex Wert`	Schreibt einen Wert in eine Zelle
`MeineMatrix get row Zeilenindex`	Liest die Werte einer Zeile. Rückgabe ist eine Liste
`MeineMatrix get column Spaltenindex`	Liest die Werte einer Spalte. Rückgabe ist eine Liste
`MeineMatrix get cell Spaltenindex Zeilenindex`	Liest den Wert einer Zelle
`MeineMatrix delete row Zeilenindex`	Löscht eine Zeile
`MeineMatrix delete column Spaltenindex`	Löscht eine Spalte
`MeineMatrix sort rows -increasing Spaltenindex`	Sortiert die Zeilen der Matrix aufsteigend gemäß einer Spalte (Achtung: ist nur für kleine Matrix geeignet)
`MeineMatrix sort rows -decreasing Spaltenindex`	Sortiert die Zeilen der Matrix absteigend gemäß einer Spalte (Achtung: ist nur für kleine Matrix geeignet)
`MeineMatrix sort columns -increasing Zeilenindex`	Sortiert die Spalten der Matrix aufsteigend gemäß einer Zeile (Achtung: ist nur für kleine Matrix geeignet)
`MeineMatrix sort columns -decreasing Zeilenindex`	Sortiert die Spalten der Matrix absteigend gemäß einer Spalte (Achtung: ist nur für kleine Matrix geeignet)
`MeineMatrix swap rows Zeilenindex1 Zeilenindex2`	Vertauscht zwei Zeilen

Tabelle 19.1: Die wichtigsten Befehle für eine Matrix (gemäß Variante 1)

Befehl	Beschreibung
`MeineMatrix swap columns Spaltenindex1 Spaltenindex2`	Vertauscht zwei Spalten
`MeineMatrix search -exact all $Suchbegriff`	Durchsucht die gesamte Matrix (Option `all`) nach einem Suchbegriff. Rückgabe ist eine Liste mit Spalten-/Zeilenpaaren für alle Treffer. Jedes Spalten-/Zeilenpaar ist seinerseits eine Liste mit Angabe des Spaltenindex und Zeilenindex. Suchoptionen sind: -exact / -glob / -regexp
`MeineMatrix search -exact row 3 $Suchbegriff`	Durchsucht die Zeile (Option `row`) nach einem Suchbegriff.
`MeineMatrix search -exact column 2 $Suchbegriff`	Durchsucht die Spalte (Option `column`) nach einem Suchbegriff.
`MeineMatrix destroy`	Löscht die Matrix und gibt den belegten Arbeitsspeicher frei
`MeineMatrix link -transpose MeinArray`	Verknüpft die Matrix mit einer Array-Variablen. Änderungen an der Matrix oder dem Array werden gegenseitig synchronisiert. Auf die Zellen kann man im Array mit `$MeinArray(Spalte,Zeile)` zugreifen. Der Befehl wird z. B. verwendet, wenn man eine Matrix mit TkTable verknüpfen will.

Listing 19.1: Matrix (Beispiel433.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require struct::matrix
4
5  ::struct::matrix MeineMatrix
6
7  MeineMatrix add columns 5
8
9  set Zeile1 {01 02 03 04 05}
10 set Zeile2 {21 22 23 24 25}
11 MeineMatrix add row $Zeile1
12 MeineMatrix add row $Zeile2
13
14 set Zeile3 {11 12 13 14 15}
15 MeineMatrix insert row 1 $Zeile3
```

```
16
17 puts "Matrix anzeigen:"
18 set Zeilen [MeineMatrix rows]
19 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
20     puts [MeineMatrix get row $Zeile]
21 }
22 set Spalten [MeineMatrix columns]
23 set Zeilen [MeineMatrix rows]
24 puts ""
25 puts "Die Matrix hat $Zeilen Zeilen und $Spalten Spalten."
26 puts "In der Zelle in Zeile 3, Spalte 4 steht:"
27 puts [MeineMatrix get cell 3 2]
28
29 puts "--------------------"
30 puts "Die Zeilen der Matrix nach Spalte 2 absteigend ↩
   sortieren:"
31 set MeineMatrix [MeineMatrix sort rows -decreasing 1]
32 set Zeilen [MeineMatrix rows]
33 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
34     puts [MeineMatrix get row $Zeile]
35 }
36 set MeineMatrix [MeineMatrix sort rows -increasing 1]
37
38 puts "--------------------"
39 puts "Die Spalten der Matrix nach Zeile 2 absteigend ↩
   sortieren:"
40 set MeineMatrix [MeineMatrix sort columns -decreasing 1]
41 set Zeilen [MeineMatrix rows]
42 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
43     puts [MeineMatrix get row $Zeile]
44 }
45 set MeineMatrix [MeineMatrix sort columns -increasing 1]
46
47 puts "--------------------"
48 puts "Die 1. und 2. Spalte der Matrix vertauschen:"
49 set MeineMatrix [MeineMatrix swap columns 0 1]
50 set Zeilen [MeineMatrix rows]
51 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
52     puts [MeineMatrix get row $Zeile]
53 }
54 set MeineMatrix [MeineMatrix swap columns 1 0]
55
56 puts "--------------------"
57 puts "Die 1. und 2. Zeile der Matrix vertauschen:"
58 set MeineMatrix [MeineMatrix swap rows 0 1]
59 set Zeilen [MeineMatrix rows]
60 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
61     puts [MeineMatrix get row $Zeile]
62 }
63 set MeineMatrix [MeineMatrix swap rows 1 0]
64
65 puts "--------------------"
66 puts "Die Spalten der Matrix anzeigen:"
```

```
67 set Spalten [MeineMatrix columns]
68 for {set Spalte 0} {$Spalte < $Spalten} {incr Spalte} {
69     puts [MeineMatrix get column $Spalte]
70 }
71
72 puts "--------------------"
73 puts "Die Zelle in Zeile 3, Spalte 4 aendern."
74 MeineMatrix set cell 3 2 999
75 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
76     puts [MeineMatrix get row $Zeile]
77 }
78
79 puts "--------------------"
80 puts "Folgende Matrix nach dem Wert 999 durchsuchen:"
81 MeineMatrix set cell 4 1 999
82 set Zeilen [MeineMatrix rows]
83 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
84     puts [MeineMatrix get row $Zeile]
85 }
86 puts ""
87 set Suchbegriff 999
88 set TrefferListe [MeineMatrix search -exact all ↩
      $Suchbegriff]
89 set AnzahlTreffer [llength $TrefferListe]
90 for {set Zaehler 0} {$Zaehler < $AnzahlTreffer} {incr ↩
      Zaehler} {
91         set Spaltenindex [lindex [lindex $TrefferListe ↩
              $Zaehler] 0]
92         set Zeilenindex [lindex [lindex $TrefferListe ↩
              $Zaehler] 1]
93         puts "Der Wert $Suchbegriff ist in Zeilenindex ↩
              $Zeilenindex und Spalteindex $Spaltenindex."
94 }
95
96 puts "--------------------"
97 MeineMatrix delete row 1
98 puts "Nachdem die 2. Zeile wurde geloescht:"
99 set Zeilen [MeineMatrix rows]
100 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
101     puts [MeineMatrix get row $Zeile]
102 }
```

```
oliver : bash — Konsole
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen
oliver@debian:~$ ./Beispiel433.tcl
Matrix anzeigen:
01 02 03 04 05
11 12 13 14 15
21 22 23 24 25

Die Matrix hat 3 Zeilen und 5 Spalten.
In der Zelle in Zeile 3, Spalte 4 steht:
24
--------------------
Die Zeilen der Matrix nach Spalte 2 absteigend sortieren:
21 22 23 24 25
11 12 13 14 15
01 02 03 04 05
--------------------
Die Spalten der Matrix nach Zeile 2 absteigend sortieren:
05 04 03 02 01
15 14 13 12 11
25 24 23 22 21
--------------------
Die 1. und 2. Spalte der Matrix vertauschen:
02 01 03 04 05
12 11 13 14 15
22 21 23 24 25
--------------------
Die 1. und 2. Zeile der Matrix vertauschen:
11 12 13 14 15
01 02 03 04 05
21 22 23 24 25
--------------------
Die Spalten der Matrix anzeigen:
01 11 21
02 12 22
03 13 23
04 14 24
05 15 25
--------------------
Die Zelle in Zeile 3, Spalte 4 aendern.
01 02 03 04 05
11 12 13 14 15
21 22 23 999 25
--------------------
Folgende Matrix nach dem Wert 999 durchsuchen:
01 02 03 04 05
11 12 13 14 999
21 22 23 999 25

Der Wert 999 ist in Zeilenindex 1 und Spalteindex 4.
Der Wert 999 ist in Zeilenindex 2 und Spalteindex 3.
--------------------
Nachdem die 2. Zeile wurde geloescht:
01 02 03 04 05
21 22 23 999 25
oliver@debian:~$
```

In Zeile 3 wird das Matrix-Paket eingebunden, so dass die Matrix-Befehle verfügbar sind. In Zeile 5 wird eine leere Matrix erzeugt. Sie hat noch keine Zeilen und keine Spalten.
In Zeile 7 werden fünf leere Spalten hinzugefügt. In den Zeilen 11 und 12 werden zwei Listen (diese enthalten die Spaltenwerte) der Matrix als neue Zeilen hinzugefügt. In Zeile 15 wird eine weitere Zeile als neue zweite Zeile (Index = 1) eingefügt.
In Zeile 18 wird die Anzahl der Zeilen ermittelt. In Zeile 20 werden die einzelnen Zeilen der Matrix ausgegeben. Die Rückgabe ist jeweils eine Liste mit den Spaltenwerten in der aktuellen Zeile. In Zeile 22 wird die Anzahl der Spalten ermittelt. In Zeile 23 wird die Anzahl der Zeilen ermittelt. In Zeile 27 wird der Inhalte der Zelle in Zeile 3 und Spalte 4 (der Zeilenindex ist 2, der Spaltenindex 3) ausgegeben.
In Zeile 31 werden die Zeilen der Matrix gemäß der zweiten Spalte absteigend sortiert. In Zeile 36 wird die Matrix aufsteigend sortiert.
In Zeile 40 werden die Spalten der Matrix gemäß der zweiten Zeile absteigend sortiert. In Zeile 45 wird die Matrix aufsteigend sortiert.
In Zeile 49 und 54 werden die erste und zweite Spalte der Matrix vertauscht.
In Zeile 58 und 63 werden die erste und zweite Spalte der Matrix vertauscht.
In Zeile 67 wird die Anzahl der Spalten bestimmt. In Zeile 69 werden die Spalten einzeln angezeigt. Die Rückgabe ist eine Liste mit den Zeilenwerten in der aktuellen Spalte.
In Zeile 74 wird der Inhalt der Zelle in Zeile 3 und Spalte 4 geändert. In Zeile 81 wird der Inhalt einer weiteren Zelle geändert.
In Zeile 88 wird die gesamte Matrix nach der Zahl 999 durchsucht. Als Rückgabe erhält man eine Liste mit den Zellen. Jede Zellenangabe ist wiederum eine Liste bestehend aus dem Spaltenindex und dem Zeilenindex. In Zeile 89 wird ermittelt, wie viele Zellen den Suchbegriff enthalten. In Zeile 90 wird die Liste mit allen gefundenen Zellen durchlaufen.
In Zeile 91 und 92 werden die Spalte bzw. Zeile der gefundenen Zelle extrahiert.
In Zeile 97 wird die zweite Zeile gelöscht.

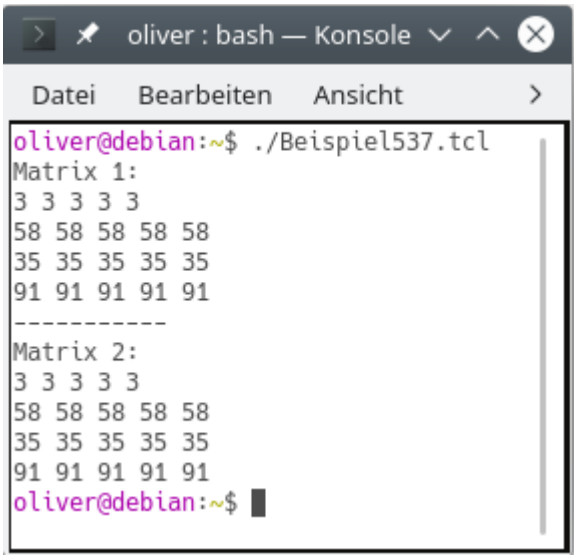
19.1 Matrix kopieren

Wenn man eine Matrix in eine andere Matrix kopieren möchte, muss man zuerst die Anzahl der Spalten in der neuen Matrix festlegen.

Listing 19.2: Matrix kopieren (Beispiel537.tcl)

```
1  #!/usr/bin/env tclsh
2  package require struct::matrix
3
4  expr srand([clock seconds])
5
6  ::struct::matrix Matrix1
7  Matrix1 add columns 5
8  for {set i 0} {$i < 4} {incr i} {
9          set tmp [expr int(100*rand())]
10         set Row [list $tmp $tmp $tmp $tmp $tmp]
11         Matrix1 add row $Row
12 }
13
14 ::struct::matrix Matrix2
15 Matrix2 add columns [Matrix1 columns]
```

```
16
17 for {set i 0} {$i < [Matrix1 rows]} {incr i} {
18         Matrix2 add row [Matrix1 get row $i]
19 }
20
21 puts "Matrix 1:"
22 set Zeilen [Matrix1 rows]
23 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
24     puts [Matrix1 get row $Zeile]
25 }
26 puts "-----------"
27 puts "Matrix 2:"
28 set Zeilen [Matrix2 rows]
29 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
30     puts [Matrix2 get row $Zeile]
31 }
```



In den Zeilen 6 bis 12 wird die `Matrix1` erstellt. Deren Inhalt soll in die `Matrix2` kopiert werden. Dazu wird in Zeile 14 die `Matrix2` erzeugt. In Zeile 15 erhält die `Matrix2` genauso viele Spalten wie die `Matrix1`. Das ist sehr wichtig, weil sonst das Kopieren nicht funktioniert. In den Zeilen 17 bis 19 wird der Inhalt der `Matrix1` zeilenweise ausgelesen und in `Matrix2` eingefügt.

19.2 Matrix mittels Liste sortieren

Man kann mit den Befehlen `MeineMatrix sort rows` bzw. `MeineMatrix sort columns` eine Matrix sortieren. Allerdings kann man nicht festlegen, ob die Matrix alphanumerisch oder numerisch sortiert werden soll. Hinzu kommt, dass der Sortier-Algorithmus sehr langsam ist und nur für kleine Matrizen geeignet ist. Das folgende Beispiel zeigt, wie man eine Matrix mit Hilfe einer Liste sortieren kann.

Listing 19.3: Matrix mittels Liste sortieren (Beispiel517.tcl)

```
#!/usr/bin/env tclsh
package require struct::matrix

expr srand([clock seconds])

::struct::matrix Matrix1
::struct::matrix Matrix2
Matrix1 add columns 5
Matrix2 add columns 5

for {set i 0} {$i < 2000} {incr i} {
        set tmp [expr int(100000*rand())]
        set Row [list $tmp $tmp $tmp $tmp $tmp]
        Matrix1 add row $Row
        Matrix2 add row $Row
}

set Sortierspalte 0

puts "Matrix1 sortieren:"
set Start [clock clicks -milliseconds]
set Matrix1 [Matrix1 sort rows -increasing $Sortierspalte]
set Zeit [expr [clock clicks -milliseconds] - $Start]
puts "Zeit: $Zeit ms"
puts "1. Zeile: [Matrix1 get row 0]"
puts "Letzte Zeile: [Matrix1 get row end]"
puts "---------------"

puts "Matrix2 mittels Liste sortieren:"
set Start [clock clicks -milliseconds]

set Liste {}
for {set i 0} {$i < [Matrix2 rows]} {incr i} {
        lappend Liste [Matrix2 get row $i]
}

if {[Matrix2 rows] > 0} {
        Matrix2 delete rows [expr [Matrix2 rows]]
}

set Liste [lsort -integer -increasing -index $Sortierspalte $Liste]

foreach Element $Liste {
        Matrix2 add row $Element
}

unset Liste

set Zeit [expr [clock clicks -milliseconds] - $Start]
puts "Zeit: $Zeit ms"
```

```
51 puts "1. Zeile: [Matrix2 get row 0]"
52 puts "Letzte Zeile: [Matrix2 get row end]"
```

```
oliver : bash — Konsole
Datei  Bearbeiten  Ansicht  Lesezeichen
oliver@debian:~$ ./Beispiel517.tcl
Matrix1 sortieren:
Zeit: 12897 ms
1. Zeile: 26 26 26 26 26
Letzte Zeile: 99949 99949 99949 99949 99949
---------------
Matrix2 mittels Liste sortieren:
Zeit: 235 ms
1. Zeile: 26 26 26 26 26
Letzte Zeile: 99949 99949 99949 99949 99949
oliver@debian:~$
```

In den Zeilen 6 bis 9 werden zwei identische Matrizen definiert und in den Zeilen 11 bis 16 mit Zufallszahlen gefüllt. In Zeile 22 wird die Matrix mit dem Matrix-eigenen Sortierbefehl sortiert, was auf meinem PC circa 12.700 Millisekunden dauert. Ab Zeile 29 wird die Matrix mittels einer Liste sortiert. In den Zeilen 33 bis 35 wird die Matrix in eine Liste umgewandelt. Jede Zeile der Matrix wird als Liste innerhalb der Variablen `Liste` gespeichert. In den Zeilen 37 bis 39 wird die Matrix geleert. In Zeile 41 wird die Liste sortiert. Hierbei kann man auch festlegen, ob die Liste alphanumerisch oder numerisch sortiert werden soll. In den Zeilen 43 bis 45 wird die Liste wieder in die Matrix umgewandelt. In Zeile 47 wird die Liste gelöscht. Der Sortiervorgang dauert jetzt nur noch ca. 240 Millisekunden.

19.3 csv-Datei in eine Matrix übernehmen

In einer Tabellenkalkulation werden folgende Daten mit einem Komma getrennt als csv-Datei gespeichert:

Beispiel.ods - LibreOffice Calc

Datei Bearbeiten Ansicht Einfügen Format Extras Daten Fenster Hilfe

Liberation Sans 10

F1

	A	B	C	D	E
1	Ueberschrift A	Ueberschrift B	Ueberschrift C	Ueberschrift D	
2	1	2,345	3;4;5	4.321,00	
3	Anton	Berta	Caesar sagte „Veni, vidi, vici"	Dora	
4	21	22	23	24	
5					

In einem Editor sieht die Datei wie folgt aus:

```
Beispiel.csv
1  Ueberschrift A,Ueberschrift B,Ueberschrift C,Ueberschrift D
2  1,"2,345",3;4;5,"4.321,00"
3  Anton,Berta,"Caesar sagte „Veni, vidi, vici“",Dora
4  21,22,23,24
5
```

Das Programm öffnet die csv-Datei, übernimmt den Inhalt in eine Matrix, zeigt die Matrix an und speichert die Matrix (jetzt mit einem Semikolon als Trennzeichen).

Listing 19.4: Matrix und csv-Datei (Beispiel432.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require csv
4  package require struct::matrix
5
6  ::struct::matrix MeineMatrix
7
8  set Dateiname Beispiel.csv
9  set f [open $Dateiname r]
10 csv::read2matrix -alternate $f MeineMatrix "," auto
11 close $f
12
13 set Zeilen [MeineMatrix rows]
14 for {set Zeile 0} {$Zeile < $Zeilen} {incr Zeile} {
15   puts [MeineMatrix get row $Zeile]
16 }
17
18 set Dateiname Ausgabe.csv
19 set f [open $Dateiname w]
20 csv::writematrix MeineMatrix $f ";"
21 close $f
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_432.tcl
{Ueberschrift A} {Ueberschrift B} {Ueberschrift C} {Ueberschrift D}
1 2,345 {3;4;5} 4.321,00
Anton Berta {Caesar sagte „Veni, vidi, vici“} Dora
21 22 23 24
oliver@debian:~$
```

In einem Editor sieht die erzeugte Datei wie folgt aus:

```
Ausgabe.csv
1   Ueberschrift A;Ueberschrift B;Ueberschrift C;Ueberschrift D
2   1;2,345;"3;4;5";4.321,00
3   Anton;Berta;Caesar sagte „Veni, vidi, vici“;Dora
4   21;22;23;24
5
```

In Zeile 3 wird das csv-Paket eingebunden. In Zeile 4 wird mit dem Befehl das Matrix-Paket eingebunden. In Zeile 6 wird eine leere Matrix erzeugt. In Zeile 10 wird der Inhalt der Datei `Beispiel.csv` in die Matrix `MeineMatrix` eingelesen. Als Trennzeichen wird das Komma festgelegt. In Zeile 13 wird die Anzahl der Zeilen der Matrix ermittelt. In den Zeilen 14 bis 16 wird jede Zeile der Matrix durchlaufen und der Inhalt angezeigt. In Zeile 20 werden die Zeilen der Matrix in eine csv-Datei geschrieben. Als Trennzeichen wird ein Semikolon verwendet.

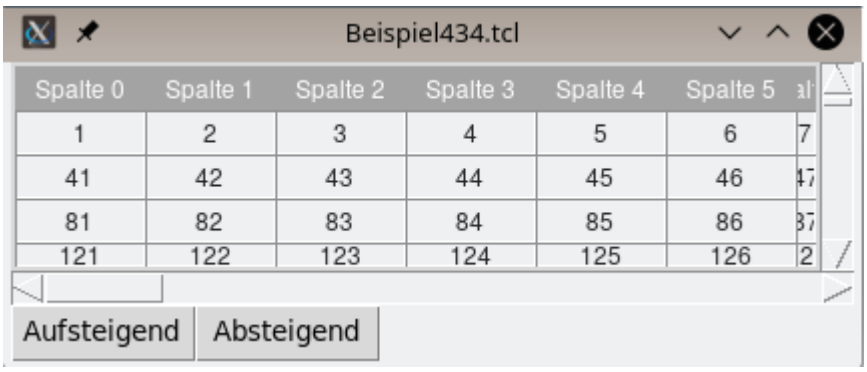
19.4 Matrix und TkTable

Das nächste Beispiel ist ein Vorgriff auf das Kapitel 50 auf Seite 563 zu TkTable, wurde aber aus inhaltlichen Gründen in dieses Kapitel integriert.

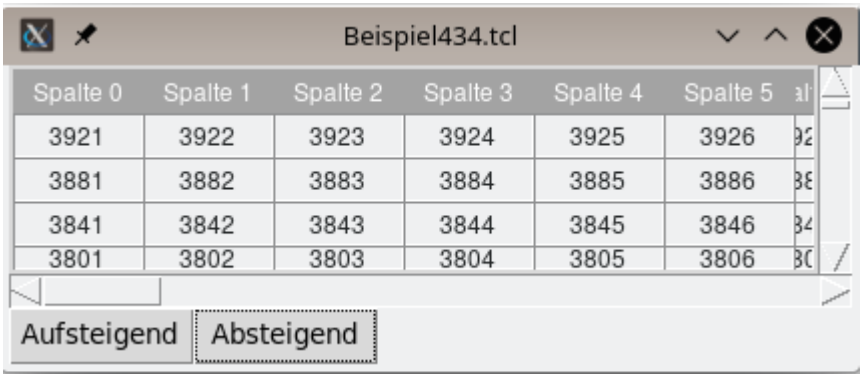
Listing 19.5: Matrix und TkTable (Beispiel434.tcl)

```
1  #!/usr/bin/env wish
2
3  package require Tktable
4  package require struct::matrix
5
6  proc Sortieren {Richtung Spalte} {
7    set Titelzeile [MeineMatrix get row 0]
8    MeineMatrix delete row 0
9    MeineMatrix sort rows $Richtung $Spalte
10   MeineMatrix insert row 0 $Titelzeile
11 }
12
13 ::struct::matrix MeineMatrix
14
15 MeineMatrix add columns 40
16
17 for {set Spalte 0} {$Spalte < 40} {incr Spalte} {
18   lappend Werte "Spalte $Spalte"
19 }
20 MeineMatrix add row $Werte
21
22 for {set Zeile 1} {$Zeile < 100} {incr Zeile} {
23   set Werte {}
24   for {set Spalte 0} {$Spalte < 40} {incr Spalte} {
25     lappend Werte [expr ($Zeile - 1) *40 + $Spalte + 1]
26   }
27   MeineMatrix add row $Werte
28 }
29
```

```
30 ttk::frame .fr
31
32 table .fr.table \
33   -rows 1 \
34   -cols 40 \
35   -titlerows 1 \
36   -titlecols 0 \
37   -height 5 \
38   -width 25 \
39   -rowheight 1 \
40   -colwidth 9 \
41   -maxheight 100 \
42   -maxwidth 400 \
43   -selectmode extended \
44   -variable Tabellenwerte \
45   -xscrollcommand {.fr.xscroll set} \
46   -yscrollcommand {.fr.yscroll set}
47
48 scrollbar .fr.xscroll -command {.fr.table xview} -orient ↩
     horizontal
49 scrollbar .fr.yscroll -command {.fr.table yview}
50
51 pack .fr -fill both -expand 1
52 pack .fr.xscroll -side bottom -fill x
53 pack .fr.yscroll -side right -fill y
54 pack .fr.table -side right -fill both -expand 1
55
56 ttk::button .btAufsteigend -text "Aufsteigend" -command {↩
     Sortieren "-increasing" 2}
57 ttk::button .btAbsteigend -text "Absteigend" -command {↩
     Sortieren "-decreasing" 2}
58 pack .btAufsteigend -side left
59 pack .btAbsteigend -side left
60
61 MeineMatrix link -transpose Tabellenwerte
62 .fr.table configure -rows [MeineMatrix rows]
```



Nach einem Klick auf den Button `Absteigend`:



In den Zeilen 17 bis 20 wird die Titelzeile (Spaltenüberschriften) der Matrix erstellt. In den Zeilen 22 bis 28 werden die weiteren Zeilen der Matrix erzeugt. In den Zeilen 30 bis 59 werden eine Tabelle mit Scroll-Leisten und zwei Buttons erzeugt (die Erklärung der einzelnen Befehle finden Sie in den Kapiteln 37 zum Geometriemanager und 50 zu TkTable). In Zeile 61 wird die Matrix `MeineMatrix` der Tabellenvariable `Tabellenwerte` zugeordnet. In Zeile 62 wird die Zeilenanzahl der Matrix ermittelt und die Zeilenanzahl der Tabelle auf den gleichen Wert gesetzt. In den Zeilen 6 bis 11 wird die Matrix und die damit verknüpfte Tabelle sortiert. Damit die Titelzeile nicht sortiert wird, wird sie in Zeile 7 gespeichert und in Zeile 8 aus der Matrix entfernt. In Zeile 9 wird die Matrix sortiert. In Zeile 10 wird die Titelzeile der Matrix wieder hinzugefügt. Da der Sortier-Befehl der Matrix langsam ist, sollte man größere Matrizes mittels Listen sortieren (siehe Kapitel 19.2 auf Seite 230)

20 Dateien und Ordner

20.1 Dateien und Ordner

Dateinamen inklusive Pfad werden in jedem Betriebssystem anders gebildet. Windows trennt z. B. die Ordner mit einem \ Strich, Linux mit einem / Strich. In Tcl/Tk verwendet man deshalb den Befehl `file` und den / Strich, um Dateien und Ordner unabhängig vom Betriebssystem anzusprechen. Grundsätzlich ist es dringend zu empfehlen, bei Ordnern und Dateinamen keine Leerzeichen zu verwenden. Für Tcl/Tk sind Leerzeichen zugleich Trennzeichen zwischen Befehlen und Optionen, so dass es zu ungewollten Fehlern im Programm kommen kann, wenn man beim Programmieren nicht sorgfältig genug ist. Nachstehend die wichtigsten Befehle im Umgang mit Dateien und Ordnern. In der Regel können die Befehle auf Dateien und Ordner gleichermaßen angewendet werden.
Die Befehle in der folgenden Tabelle sind teilweise sehr lang, so dass sie mehrzeilig dargestellt werden. Im Programm ist der Befehl jedoch immer einzeilig zu schreiben.

Tabelle 20.1: Die wichtigsten Befehle für Dateien und Ordner

Befehl	Beschreibung
`puts [pwd]`	Aktueller Arbeitsordner (pwd = print working directory)
`cd /tmp`	Wechselt den Arbeitsordner
`puts $env(HOME)`	Der Home-Ordner des Benutzers
`puts [info nameofexecutable]`	Der Name des Programms, das das tcl-Skript ausführt
`puts [info script]`	Der Name des tcl-Skripts (inkl. relativer Pfadangabe)
`puts [file volumes]`	Bei Linux das Wurzelverzeichnis, bei Windows alle Laufwerke C, D usw.
`puts [file readlink $Datei]`	Wenn die Datei ein symbolischer Link ist, wird der Name der Datei zurückgegeben, auf den der Link verweist
`set Ordner [file join home user]`	Bildet einen Pfad
`set DateinameMitPfad [file join home user Datei.txt]`	Bildet einen Dateinamen mit Pfad
`set DateinameMitPfad [file join $Liste]`	Bildet einen Dateinamen mit Pfad aus den Elementen der Liste

Tabelle 20.1: Die wichtigsten Befehle für Dateien und Ordner

Befehl	Beschreibung
`puts [file split /home/user/Datei.txt]`	Trennt den Dateinamen in die einzelnen Elemente
`set Liste [file split /home/user/Datei.txt]`	Trennt den Dateinamen in die einzelnen Elemente und speichert sie in einer Liste
`puts [file dirname /home/user/Datei.txt]`	Extrahiert den Namen des Ordners
`puts [file tail /home/user/Datei.txt]`	Extrahiert den Dateinamen
`puts [file rootname /home/user/Datei.txt]`	Extrahiert den Pfad mit Dateinamen, aber ohne die Dateiendung (Extension)
`puts [file extension /home/user/Datei.txt]`	Extrahiert die Dateiendung (Extension)
`puts [file exist /home/user/Datei.txt]`	Prüft, ob die Datei existiert (1 = ja, 0 = nein).
`puts [file exist /home/user]`	Prüft, ob der Ordner existiert (1 = ja, 0 = nein).
`puts [file type /home/user/Datei.txt]`	Ermittelt den Typ der Datei. Rückgabewerte sind: file (= Datei) directory (= Ordner) link (= Link)
`puts [file size /home/user/Datei.txt]`	Gibt die Dateigröße in Bytes an
`file mkdir $Ordner`	Erstellt einen Ordner. Wenn die übergeordneten Ordner noch nicht existieren, werden diese ebenfalls erzeugt.
`file delete /home/user/Datei.txt`	Löscht die Datei
`file delete -force /home/user`	Löscht den Ordner. Ein Ordner, der noch Dateien enthält wird nicht gelöscht. Die Option `-force` bewirkt, dass auch Ordner gelöscht werden, die noch Dateien enthalten.

Tabelle 20.1: Die wichtigsten Befehle für Dateien und Ordner

Befehl	Beschreibung
`file rename -force /home/user/DateiAlt.txt /home/user/DateiNeu.txt`	Benennt eine Datei bzw. einen Ordner um oder verschiebt die Datei bzw. den Ordner. Wenn der neue Dateiname bereits existiert, wird die Datei nicht umbenannt oder verschoben. Die Option `-force` bewirkt, dass die Datei auch dann umbenannt bzw. verschoben wird, wenn der neue Dateiname bereits existiert. Die alte Datei wird somit überschrieben.
`file copy -force /home/user/Datei.txt /home/user/tmp/Datei.txt`	Kopiert eine Datei oder einen Ordner in einen Zielordner. Wenn die Datei bereits im Zielordner existiert, wird die Datei nicht kopiert. Die Option `-force` bewirkt, dass die Datei auch dann kopiert wird, wenn der neue Dateiname bereits existiert. Die alte Datei wird somit überschrieben.
`puts [file isdirectory /home/user/]`	Prüft, ob es sich um einen Ordner handelt (1 = ja, 0 = nein).
`puts [file isfile /home/user/Datei.txt]`	Prüft, ob es sich um eine Datei handelt (1 = ja, 0 = nein).
`puts [file readable /home/user/Datei.txt]`	Prüft, ob der Anwender Leserechte auf die Datei bzw. in dem Ordner hat (1 = ja, 0 = nein).
`puts [file writable /home/user/Datei.txt]`	Prüft, ob der Anwender Schreibrechte auf die Datei bzw. in dem Ordner hat (1 = ja, 0 = nein).
`puts [file executable /home/user/Datei]`	Prüft, ob der Anwender Ausführungsrechte auf die Datei bzw. auf den Ordner hat (1 = ja, 0 = nein).
`puts [file mtime /home/user/Datei.txt]`	Zeit der letzten Änderung am Datei-Inhalt (in Sekunden seit dem 1.1.1970 0:00 Uhr).
`puts [file atime /home/user/Datei.txt]`	Zeit des letzten Zugriffs (in Sekunden seit Beginn der Zeitzählung des jeweiligen Betriebssystems, bei Linux z. B. 1.1.1970).

Tabelle 20.1: Die wichtigsten Befehle für Dateien und Ordner

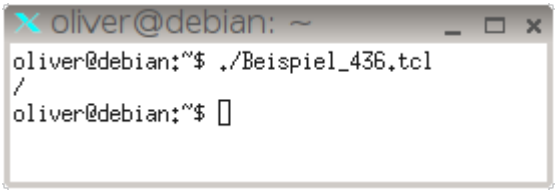
Befehl	Beschreibung
`puts [file ctime /home/user/Datei.txt]`	Zeit der letzten Statusänderung (in Sekunden seit Beginn der Zeitzählung des jeweiligen Betriebssystems, bei Linux z. B. 1.1.1970).
`puts [file gid /home/user/Datei.txt]`	Group-ID (nur Linux)
`puts [file uid /home/user/Datei.txt]`	User-ID (nur Linux)

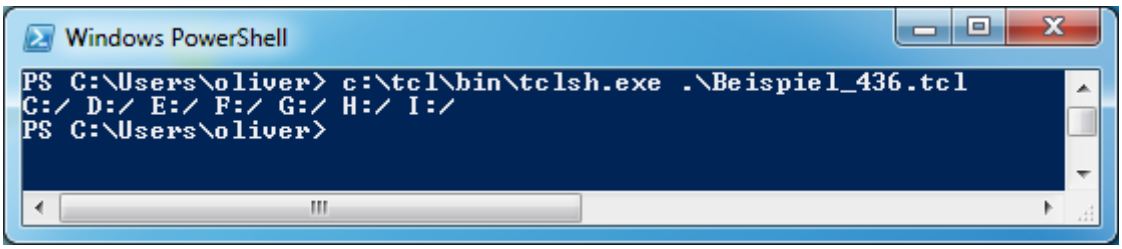
Listing 20.1: Skriptname, Home-Ordner, aktueller Ordner (Beispiel132.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts "Der Name des ausfuehrenden Programms: [info ↩
    nameofexecutable]"
4 puts "Der Name des Skripts: [info script]"
5 puts "Der Home-Ordner: $env(HOME)"
6 puts "Der aktuelle Arbeitsordner: [pwd]"
```

Listing 20.2: Wurzelverzeichnis (Linux) bzw. Laufwerke (Windows) (Beispiel436.tcl)

```
1 #!/usr/bin/env tclsh
2
3 puts [file volumes]
```





Listing 20.3: Dateinamen bilden (Beispiel133.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Dateiname [file join $env(HOME) MeineDatei.txt]
4  puts "Dateiname 1: $Dateiname"
5
6  set Dateiname [file join [file dirname [info script]] ↩
       MeineDatei.txt]
7  puts "Dateiname 2: $Dateiname"
```

In Zeile 3 bzw. Zeile 6 werden verschiedene Dateinamen gebildet. Da man nicht weiß, welche Ordner auf dem ausführenden System vorhanden sind, muss man entweder die Ordner selbst anlegen oder verwendet Ordner, die immer vorhanden sind: das sind der Home-Ordner sowie der Ordner, in dem das tcl-Skript abgelegt ist.

Listing 20.4: Teile des Dateinamens (Beispiel134.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Dateiname [file join $env(HOME) MeineDatei.txt]
4  puts "Dateiname: $Dateiname"
5  puts "Die einzelnen Bestandteile sind: [file split ↩
       $Dateiname]"
6  puts "Nur der Ordner: [file dirname $Dateiname]"
7  puts "Nur der Dateiname (ohne Ordner): [file tail ↩
       $Dateiname]"
8  puts "Nur die Dateiendung: [file extension $Dateiname]"
9  puts "Dateiname mit Ordner, aber ohne Dateiendung: [file ↩
       rootname $Dateiname]"
10 puts "Dateiname ohne Ordner und ohne Dateiendung: [file ↩
       tail [file rootname $Dateiname]]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_134.tcl
Dateiname: /home/oliver/MeineDatei.txt
Die einzelnen Bestandteile sind: / home oliver MeineDatei.txt
Nur der Ordner: /home/oliver
Nur der Dateiname (ohne Ordner): MeineDatei.txt
Nur die Dateiendung: .txt
Dateiname mit Ordner, aber ohne Dateiendung: /home/oliver/MeineDatei
Dateiname ohne Ordner und ohne Dateiendung: MeineDatei
oliver@debian:~$
```

Listing 20.5: Datei-Informationen (Beispiel135.tcl)

```
#!/usr/bin/env tclsh

set Datei [info script]
puts "Datei: $Datei"

if {[file exist $Datei]} {
  puts "Die Datei $Datei existiert."
} else {
  puts "Die Datei $Datei existiert nicht."
}

if {[file isdirectory $Datei]} {
  puts "Die Datei $Datei ist ein Ordner."
} else {
  puts "Die Datei $Datei ist kein Ordner."
}
if {[file isfile $Datei]} {
  puts "Die Datei $Datei ist eine Datei."
} else {
  puts "Die Datei $Datei ist keine Datei."
}
puts "Die Datei ist vom Typ [file type $Datei]"
puts "Dateigroesse in Bytes: [file size $Datei]"
set Zeit [file mtime $Datei]
puts "Letzte Aenderung: [clock format $Zeit -format {%d.%m.%Y %H:%M:%S}]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_135.tcl
Datei: ./Beispiel_135.tcl
Die Datei ./Beispiel_135.tcl existiert.
Die Datei ./Beispiel_135.tcl ist kein Ordner.
Die Datei ./Beispiel_135.tcl ist eine Datei.
Die Datei ist vom Typ file
Dateigröße in Bytes: 656
Letzte Änderung: 19.07.2014 20:29:32
oliver@debian:~$
```

In Zeile 24 wird die Zeit der letzten Änderung an der Datei ermittelt. Da diese Zeit in Sekunden seit dem 1.1.1970 0:00 Uhr gemessen wird, ist sie für einen Menschen zunächst nicht interpretierbar. In Zeile 25 wird die Zeit deshalb formatiert ausgegeben.

Listing 20.6: Ordner und Datei erstellen, kopieren, umbenennen, löschen (Beispiel136.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc ExistenzPruefen {Pfad} {
4    if {[file exist $Pfad]} {
5      puts "Die Datei bzw. der Ordner $Pfad existiert."
6    } else {
7      puts "Die Datei bzw. der Ordner $Pfad existiert nicht.
       "
8    }
9  }
10
11 set Datei [info script]
12 set Ordner [file join $env(HOME) Test]
13
14 puts "Den Ordner $Ordner erstellen:"
15 file mkdir $Ordner
16 ExistenzPruefen $Ordner
17 puts ""
18
19 puts "Die Datei $Datei in den Ordner $Ordner kopieren:"
20 file copy $Datei $Ordner
21 set DateiNeu [file join $Ordner [file tail $Datei]]
22 ExistenzPruefen $DateiNeu
23 puts ""
24
25 set DateiNeu2 [file join $Ordner Neu.txt]
26 puts "Die Datei $DateiNeu umbenennen in $DateiNeu2:"
27 file rename $DateiNeu $DateiNeu2
28 ExistenzPruefen $DateiNeu2
29 puts ""
30
31 puts "Die Datei $DateiNeu2 loeschen:"
32 file delete $DateiNeu2
33 ExistenzPruefen $DateiNeu2
34 puts ""
35
36 puts "Den Ordner $Ordner loeschen:"
37 file delete $Ordner
38 ExistenzPruefen $Ordner
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_136.tcl
Den Ordner /home/oliver/Test erstellen:
Die Datei bzw. der Ordner /home/oliver/Test existiert.

Die Datei ./Beispiel_136.tcl in den Ordner /home/oliver/Test kopieren:
Die Datei bzw. der Ordner /home/oliver/Test/Beispiel_136.tcl existiert.

Die Datei /home/oliver/Test/Beispiel_136.tcl umbenennen in /home/oliver/Test/Neu.t
Die Datei bzw. der Ordner /home/oliver/Test/Neu.txt existiert.

Die Datei /home/oliver/Test/Neu.txt löschen:
Die Datei bzw. der Ordner /home/oliver/Test/Neu.txt existiert nicht.

Den Ordner /home/oliver/Test löschen:
Die Datei bzw. der Ordner /home/oliver/Test existiert nicht.
oliver@debian:~$
```

In den Zeilen 3 bis 8 wird eine Prozedur definiert, die prüft, ob eine Datei bzw. ein Ordner existiert. In Zeile 11 wird der Dateiname des tcl-Skripts ermittelt. In Zeile 12 wird ein neuer Ordnername erstellt. In der Zeile 15 wird ein neuer Ordner im Home-Ordner erstellt. In Zeile 20 wird die Skriptdatei in den neuen Ordner kopiert. In Zeile 21 wird der Name der kopierten Datei erstellt. In Zeile 27 wird die Datei umbenannt. In Zeile 32 wird die Datei gelöscht. In Zeile 37 wird der Ordner gelöscht.

20.2 Ordner durchsuchen mit glob

Befehl:

- `[glob -nocomplain -directory Startordner -types {Dateitypen Dateieigenschaften} Suchbegriff]`

Der `glob`-Befehl durchsucht einen Ordner nach Dateien und Unterordnern. Er durchsucht nur den angegebenen Ordner, nicht die Unterordner. Der Befehl gibt die gefundenen Dateien oder Unterordner als Liste zurück. Die Option `-nocomplain` verhindert, dass es zu einer Fehlermeldung kommt, wenn der `glob`-Befehl keine Datei bzw. keinen Unterordner findet.

Tabelle 20.2: Die wichtigsten Dateitypen

Dateityp	Beschreibung
`f`	Datei (File)
`d`	Ordner (Directory)
`l`	Symbolischer Link (Link)

Es können mehrere Dateitypen übergeben werden. Beispiel: f d sucht nach Dateien und Ordnern.

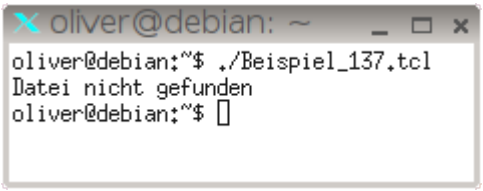
Tabelle 20.3: Dateieigenschaften

Dateieigenschaften	Beschreibung
`r`	Datei/Ordner darf gelesen werden (Read)
`w`	Datei/Ordner darf geschrieben werden (Write)
`x`	Datei/Ordner darf ausgeführt werden (Execute)
`hidden`	Datei/Ordner ist verborgen
`{f d hidden}`	sucht nur nach verborgenen Dateien und Ordnern
`{f rwx}`	sucht nach Dateien, die sowohl gelesen als auch geschrieben und ausgeführt werden können. Alle drei Dateieigenschaften müssen erfüllt sein.
`{f rw}`	sucht nach Dateien, die sowohl gelesen als auch geschrieben werden können. Dabei ist es nicht relevant, ob die Datei auch ausgeführt werden kann.

Als Suchbegriff kann man einen Datei- oder Ordnernamen eingeben bzw. Teile davon. Als Platzhalter können `*` für beliebig viele Zeichen oder `?` für genau ein Zeichen verwendet werden.

Listing 20.7: Prüft, ob eine bestimmte Datei vorhanden ist (Beispiel137.tcl)

```
1 #!/usr/bin/env tclsh
2
3 if {[glob -nocomplain -types f MeineDatei.txt] eq ""} {
4   puts "Datei nicht gefunden"
5 } else {
6   puts "Datei gefunden"
7 }
```



Listing 20.8: Listet alle Dateien in einem Ordner auf (Beispiel138.tcl)

```
1 #!/usr/bin/env tclsh
2
```

```
3 set Ordner [file dirname [info script]]
4 set Liste {}
5 if { [catch {glob -nocomplain -directory $Ordner -types {f}
    } *} tmp] == 0 } {
6   set Liste $tmp
7 }
8 puts $Liste
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_138.tcl
./Beispiel_131.tcl ./Beispiel_132.tcl ./Beispiel_133.tcl ./Beispiel_134.
tcl ./Beispiel_135.tcl ./Beispiel_136.tcl ./Beispiel_137.tcl ./Beispiel_
138.tcl ./Beispiel_139.tcl ./Beispiel_140.tcl
oliver@debian:~$
```

In Zeile 5 wird durch die Option `-types {f}` nach Dateien gesucht. Die gefundenen Dateien werden zunächst in der Variablen `tmp` gespeichert. Der ganze `glob`-Befehl ist sicherheitshalber in einen `catch`-Befehl (siehe Kapitel 22 auf Seite 265) gesetzt, um bei Bedarf die Fehlermeldung abzufangen, falls es z. B. den Ordner nicht gibt. Wenn der `glob`-Befehl erfolgreich ausgeführt wurde, liefert der `catch`-Befehl als Rückgabewert `0`. Nur dann werden die gefundenen Dateien in der Variablen `Liste` gespeichert (Zeile 6).

Listing 20.9: Ohne Rekursion: Alle Dateien und Ordner inklusive Unterordner auflisten (ohne versteckte Ordner und Dateien) (Beispiel139.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Ordner {} ; # Liste mit allen Ordner
4 set Dateien {} ; # Liste mit allen Dateien
5 set AktuellerOrdner ""
6
7 lappend Ordner [file join $env(HOME) test] ; # Startordner
8 set Index 0 ; # Startwert, damit die while-Schleife
    mindestens einmal durchlaufen wird
9
10 while {$Index < [llength $Ordner] } {
11   set AktuellerOrdner [lindex $Ordner $Index]
12
13   # Nicht die Ordner . bzw .. aufrufen
14   if {[file tail $AktuellerOrdner] ni {. ..}} {
15     if {[catch {glob -nocomplain -directory
        $AktuellerOrdner -types {f} *} tmp] == 0 } {
16       foreach Element $tmp {
17         if {[file tail $Element] ni {. ..}} {
18           lappend Dateien $Element
19         }
20       }
21         }
22     if {[catch {glob -nocomplain -directory
        $AktuellerOrdner -types {d} *} tmp] == 0 } {
23       foreach Element $tmp {
```

```
24          if {[file tail $Element] ni {. ..}} {
25            lappend Ordner $Element
26          }
27        }
28      }
29    }
30    incr Index
31  }
32  puts "Ordner:"
33  foreach Element $Ordner {
34    puts $Element
35  }
36  puts ""
37  puts "Dateien:"
38  foreach Element $Dateien {
39    puts $Element
40  }
```

```
oliver@debian: ~/Test
oliver@debian:~/Test$ ./Beispiel_139.tcl
Ordner:
/home/oliver/Test
/home/oliver/Test/Ordner2
/home/oliver/Test/Ordner1
/home/oliver/Test/Ordner1/Unterordner1-1
/home/oliver/Test/Ordner1/Unterordner1-2

Dateien:
/home/oliver/Test/Beispiel_139.tcl
/home/oliver/Test/Datei1.tcl
/home/oliver/Test/Datei2.tcl
/home/oliver/Test/Ordner2/Datei7.tcl
/home/oliver/Test/Ordner1/Datei3.tcl
/home/oliver/Test/Ordner1/Unterordner1-1/Datei4.tcl
/home/oliver/Test/Ordner1/Unterordner1-1/Datei5.tcl
/home/oliver/Test/Ordner1/Unterordner1-2/Datei6.tcl
oliver@debian:~/Test$
```

Listing 20.10: Mit Rekursion: Alle Dateien und Ordner inklusive Unterordner auflisten (mit versteckten Ordnern und Dateien) (Beispiel438.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc DateienFinden {Startordner} {
4    set Liste ""
5
6    # Dateien
7    foreach Element [glob -nocomplain -types f -directory ↩
       $Startordner *] {
8      if {[file tail $Element] ni {. ..}} {
9            lappend Liste $Element
10         }
11   }
```

```
12
13    foreach Element [glob -nocomplain -types {f hidden} ↩
        -directory $Startordner *] {
14      if {[file tail $Element] ni {. ..}} {
15        lappend Liste $Element
16      }
17    }
18
19    # Ordner
20    foreach Element [glob -nocomplain -types d -directory ↩
        $Startordner *] {
21      if {[file tail $Element] ni {. ..}} {
22        lappend Liste $Element
23        set Dateien [DateienFinden $Element]
24        set Liste [concat $Liste $Dateien]
25      }
26    }
27
28    foreach Element [glob -nocomplain -types {d hidden} ↩
        -directory $Startordner *] {
29      if {[file tail $Element] ni {. ..}} {
30        lappend Liste $Element
31        set Dateien [DateienFinden $Element ]
32        set Liste [concat $Liste $Dateien]
33      }
34    }
35
36    set Liste [lsort -dictionary -unique $Liste]
37    return $Liste
38  }
39
40  set Startordner [file join $env(HOME) test]
41  set Liste [DateienFinden $Startordner]
42  foreach Element $Liste {
43    puts $Element
44  }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_438.tcl
/home/oliver/test/Datei1.txt
/home/oliver/test/Datei2.txt
/home/oliver/test/Ordner1
/home/oliver/test/Ordner1/Datei3.txt
/home/oliver/test/Ordner1/Datei4.txt
/home/oliver/test/Ordner1/Datei5.txt
/home/oliver/test/Ordner2
/home/oliver/test/Ordner2/Datei6.txt
oliver@debian:~$
```

In Zeile 11 werden durch die Option `hidden` die versteckten Dateien gesucht. Ebenso werden in der Zeile 22 die versteckten Ordner ermittelt. Die Zeile 23 prüft, dass es sich nicht um die Ordner `.` bzw. `..` handelt. In den Zeilen 18 und 25 ruft die Prozedur sich

selbst auf (mit einem neuen Startordner). Das nennt man eine Rekursion.

20.3 Ordner durchsuchen mit fileutil::find

Befehle:

- `package require fileutil`
- `set Dateien [fileutil::find $Ordner Vergleich]`

Die Bibliothekt `tcllib`, die standardmäßig installiert ist, enthält u. a. das Paket `fileutil`. Dieses Paket stellt den Befehl `fileutil::find` zur Verfügung, mit dem man den aktuellen Ordner inklusive aller Unterordner nach Dateien und Ordner durchsuchen kann. Mit Hilfe einer Vergleichsprozedur kann man festlegen, welche Dateien bzw. Ordner gefunden werden sollen.

Listing 20.11: Aktuellen Ordner inklusive Unterordner nach Dateien durchsuchen (Beispiel367.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require fileutil
4
5  proc Vergleich {Dateiname} {
6    return [string match *.tcl $Dateiname]
7  }
8
9  set Ordner [file dirname [info script]]
10 set Dateien [fileutil::find $Ordner Vergleich]
11 foreach Datei $Dateien {
12         puts $Datei
13 }
```

```
oliver@debian: ~/Test
oliver@debian:~/Test$ ./Beispiel_367.tcl
./Beispiel_367.tcl
./Beispiel_370.tcl
./Datei1.tcl
./Datei2.tcl
./Ordner1/Datei3.tcl
./Ordner2/Datei7.tcl
./Ordner1/Unterordner1-1/Datei4.tcl
./Ordner1/Unterordner1-1/Datei5.tcl
./Ordner1/Unterordner1-2/Datei6.tcl
oliver@debian:~/Test$
```

In den Zeilen 5 bis 7 wird eine Prozedur definiert, die festlegt, welche Dateien gefunden werden sollen. Der Rückgabewert ist `1`, wenn die Datei auf das Suchmuster passt, sonst `0`. In Zeile 10 wird der Befehl `fileutil::find` aufgerufen. Als Parameter werden der Startordner und die Prozedur übergeben. Der Befehl durchsucht nun die Ordner nach allen Dateien und übergibt jeden einzelnen Dateinamen an die Prozedur `Vergleich`. Die

Prozedur überprüft, ob der Dateiname auf `.tcl` endet und liefert als Rückgabewert den Wert `1` (Dateiname passt) oder `0` (Dateiname passt nicht). Wenn der Rückgabewert `1` ist, wird der Dateiname zu der Variablen `Dateien` hinzugefügt. Die Pfadangabe der Dateien ist dabei relativ zum Startordner.

Listing 20.12: Komplette Pfad-Angabe in der Vergleichsprozedur (Beispiel399.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require fileutil
4
5  proc Vergleich {Dateiname} {
6    puts "zu pruefende Datei: $Dateiname"
7    puts "Dateiname mit Pfad: [file join [pwd] $Dateiname]"
8    return [string match *.txt $Dateiname]
9  }
10
11 set Ordner [file dirname [info script]]
12 set Dateien [fileutil::find $Ordner Vergleich]
13 puts "Ergebnis:"
14 foreach Datei $Dateien {
15         puts $Datei
16 }
```

```
oliver@debian: ~/Test
oliver@debian:~/Test$ ./Beispiel_399.tcl
zu prüfende Datei: Beispiel_399.tcl
Dateiname mit Pfad: /home/oliver/Test/Beispiel_399.tcl
zu prüfende Datei: Text1.txt
Dateiname mit Pfad: /home/oliver/Test/Text1.txt
zu prüfende Datei: Ordner1
Dateiname mit Pfad: /home/oliver/Test/Ordner1
zu prüfende Datei: Text2.txt
Dateiname mit Pfad: /home/oliver/Test/Ordner1/Text2.txt
Ergebnis:
./Text1.txt
./Ordner1/Text2.txt
oliver@debian:~/Test$
```

In Zeile 6 wird die zu prüfende Datei angezeigt. Der Dateiname ist ohne Pfadangabe. In Zeile 7 wird der Dateiname inklusive Pfad angezeigt. Dazu wird der Dateiname mit dem Befehl `[file join [pwd] $Dateiname]` um den Pfad erweitert. In Zeile 15 werden alle Dateien, die dem Vergleichskriterium `*.txt` entsprachen, angezeigt.

Listing 20.13: Alle Unterordner auflisten (Beispiel370.tcl)

```
#!/usr/bin/env tclsh

package require fileutil

proc Vergleich {Dateiname} {
  if {[file type $Dateiname] == "directory"} {
    return 1
```

```
8     } else {
9       return 0
10    }
11  }
12
13  set Ordner [file dirname [info script]]
14  set Dateien [fileutil::find $Ordner Vergleich]
15  foreach Datei $Dateien {
16          puts $Datei
17  }
```

```
oliver@debian: ~/Test
oliver@debian:~/Test$ ./Beispiel_370.tcl
./Ordner1
./Ordner2
./Ordner1/Unterordner1-1
./Ordner1/Unterordner1-2
oliver@debian:~/Test$
```

In Zeile 6 wird geprüft, ob es sich um einen Ordner handelt.

20.4 Textdatei speichern

Befehl:

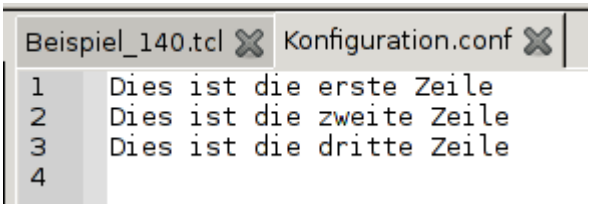
```
set f [open Dateiname w]
puts $f "Text"
close $f
```

Oft soll das Programm bestimmte Daten dauerhaft speichern, damit diese Daten beim nächsten Programmstart wieder zur Verfügung stehen. Klassische Beispiele sind Konfigurationsdaten oder Spielstände. Die Daten werden als reine Textdateien gespeichert, so dass man sie mit jedem Editor bearbeiten kann. Denken Sie daran, bei Ordner- und Dateinamen möglichst keine Leerzeichen zu verwenden.

Listing 20.14: Textdatei erstellen (Beispiel140.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Skriptname [info script]
4  set Dateiname [file join [file dirname $Skriptname] ↩
       Konfiguration.conf]
5  set f [open $Dateiname w]
6  puts $f "Dies ist die erste Zeile"
7  puts $f "Dies ist die zweite Zeile"
8  puts $f "Dies ist die dritte Zeile"
9  close $f
```

Öffnen Sie mit einem Texteditor die Datei `Konfiguration.conf`.



In Zeile 3 wird der Dateiname des Tcl-Programms ermittelt. In Zeile 4 wird der Dateiname (inkl. Pfad) für die Textdatei gebildet. Die Datei wird in demselben Ordner gespeichert in dem auch das Tcl-Programm abgelegt ist. In Zeile 5 wird die Datei zum Schreiben geöffnet. In den Zeilen 6 bis 8 werden drei Textzeilen in die Datei geschrieben. In Zeile 9 wird die Datei geschlossen.

20.5 Textdatei einlesen

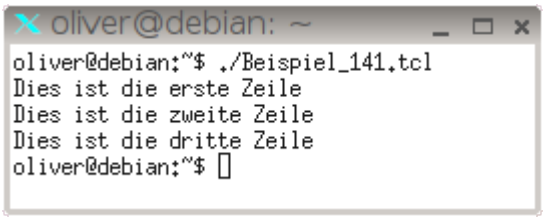
Befehl:

```
set f [open Dateiname r]
while {[gets $f Zeile] >= 0} {
  puts $Zeile
}
close $f
```

Das Einlesen einer Textdatei funktioniert so:

Listing 20.15: Textdatei einlesen (Beispiel141.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Skriptname [info script]
4  set Dateiname [file join [file dirname $Skriptname] ↩
     Konfiguration.conf]
5  if {[file exist $Dateiname] == 0} {exit}
6  set f [open $Dateiname r]
7  while {[gets $f Zeile] >= 0} {
8    puts $Zeile
9  }
10 close $f
```



In Zeile 3 wird der Dateiname des Tcl-Programms ermittelt. In Zeile 4 wird der Dateiname (inkl. Pfad) für die Textdatei gebildet. Sie muss in demselben Ordner abgelegt sein, in dem auch das Tcl-Programm gespeichert ist. In Zeile 5 wird geprüft, ob die Datei existiert. Wenn nicht, wird das Programm beendet. In Zeile 6 wird die Datei zum Lesen geöffnet. In den Zeile 7 bis 9 wird die Datei zeilenweise gelesen. In Zeile 10 wird die Datei geschlossen.

20.6 Konfigurationsdatei speichern und einlesen

Befehle:

- `array get`
- `source`
- `dict for`
- `dict set`

Wenn man die Konfigurationsdaten in einem Array oder einem Dictionary speichert, kann man die Daten sehr einfach in einer Textdatei speichern und wieder einlesen.

Listing 20.16: Konfigurationsdaten (Array) (Beispiel371.tcl)

```
#!/usr/bin/env tclsh

proc KonfigurationSpeichern {} {
  global Glob

  set f [open Konf.txt w]
  puts $f "array set Glob [list [array get Glob]]"
  close $f
}

proc KonfigurationLaden {} {
  global Glob

  source Konf.txt
}

array set Glob {}
set Glob(Zahl) 100
set Glob(Text) "abc"

puts "Glob(Zahl): $Glob(Zahl)"
puts "Glob(Text): $Glob(Text)"
puts "----------"

KonfigurationSpeichern

set Glob(Zahl) 0
set Glob(Text) ""

puts "Glob(Zahl): $Glob(Zahl)"
puts "Glob(Text): $Glob(Text)"
puts "----------"

KonfigurationLaden

puts "Glob(Zahl): $Glob(Zahl)"
puts "Glob(Text): $Glob(Text)"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_371.tcl
Glob(Zahl): 100
Glob(Text): abc
----------
Glob(Zahl): 0
Glob(Text):
----------
Glob(Zahl): 100
Glob(Text): abc
oliver@debian:~$
```

In den Zeilen 17 bis 19 wird ein Array definiert, das als globale Variable dienen soll. In den Zeilen 21 und 22 wird der Inhalt des Arrays angezeigt. In Zeile 25 wird die Prozedur `KonfigurationSpeichern` aufgerufen, die den Inhalt des Arrays in eine Textdatei speichert. In den Zeilen 27 und 28 wird das Array geleert. In den Zeilen 30 und 31 wird das leere Array ausgegeben. In Zeile 34 wird durch die Prozedur `KonfigurationLaden` das Array mit den ursprünglichen Werten wieder befüllt. In den Zeile 36 und 37 wird der Inhalt des befüllten Arrays angezeigt. In den Zeilen 3 bis 9 wird das Array in eine Textdatei gespeichert. Entscheidend ist die Zeile 7: Der Befehl `[array get Glob]` liest den Inhalt des Arrays `Glob` aus. Der vorangestellte `list`-Befehl stellt sicher, dass der Array-Inhalt zu einem gültigen Tcl-Befehl wird. Der nochmals vorangestellte Text `array set Glob` vervollständigt den gesamten Befehl. Im Ergebnis sieht der Befehl wie folgt aus:

```
array set Glob {Text abc Zahl 100}
```

Dieser Befehl wird dann in der Textdatei `Konf.txt` gespeichert. In der Zeile 14 wird mit dem Befehl `source` die Textdatei `Konf.txt` eingelesen und der Inhalt sofort als Tcl-Befehl ausgeführt.

Listing 20.17: Konfigurationsdaten (Dictionary) (Beispiel429.tcl)

```
1  #!/usr/bin/env tclsh
2
3  proc KonfigurationSpeichern {Daten} {
4          set f [open Konf.txt w]
5          dict for {Schluessel Wert} $Daten {puts $f "
               $Schluessel:$Wert"}
6          close $f
7  }
8
9  proc KonfigurationLaden {} {
10         set f [open Konf.txt r]
11         while {[gets $f Zeile] >= 0} {
12                 set tmp [string first ":" $Zeile]
13                 set Schluessel [string range $Zeile 0 [
                       expr $tmp - 1]]
14                 set Wert [string range $Zeile [expr $tmp +
                       1] end]
15                 dict set Daten $Schluessel $Wert
16         }
17         close $f
```

```
18          return $Daten
19 }
20
21 dict set Konfiguration Zahl 100
22 dict set Konfiguration Text "abc"
23
24 puts "Zahl: [dict get $Konfiguration Zahl]"
25 puts "Text: [dict get $Konfiguration Text ]"
26 puts "----------"
27
28 KonfigurationSpeichern $Konfiguration
29
30 dict set Konfiguration Zahl 0
31 dict set Konfiguration Text ""
32
33 puts "Zahl: [dict get $Konfiguration Zahl]"
34 puts "Text: [dict get $Konfiguration Text ]"
35 puts "----------"
36
37 set Konfiguration [KonfigurationLaden]
38
39 puts "Zahl: [dict get $Konfiguration Zahl]"
40 puts "Text: [dict get $Konfiguration Text ]"
```

In den Zeilen 21 und 22 wird ein Dictionary definiert. In den Zeilen 3 bis 7 wird das Dictionary in eine Textdatei gespeichert. Dazu wird der Befehl `dict for` verwendet, der alle Schlüssel-Wert-Paare aus dem Dictionary ausliest. Beim Speichern in die Textdatei wird der Schlüssel mit einem Doppelpunkt vom Wert abgetrennt. In den Zeilen 9 bis 19 wird die Textdatei eingelesen. Dazu wird jede Zeile einzeln eingelesen und nach dem ersten Doppelpunkt gesucht. Der erste Doppelpunkt trennt den Schlüssel vom Wert. Danach werden Schlüssel und Wert in ein lokales Dictionary gespeichert. In Zeile 18 wird das lokale Dictionary an den aufrufenden Programmteil zurückgegeben. In Zeile 37 wird das aus der Prozedur zurückgegebene Dictionary in das globale Dictionary Konfiguration gespeichert.

20.7 csv-Dateien öffnen bzw. erstellen

Befehle:

- `package require csv`

- `::csv::split $Zeile ","`
- `::csv::split -alternate $Zeile ","`
- `::csv::join $Liste ","`

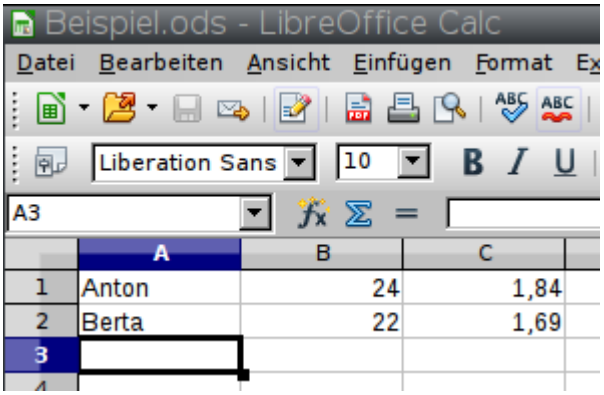
csv-Dateien sind grundsätzlich Textdateien. Allerdings werden die Daten in der csv-Datei durch ein Trennzeichen voneinander abgegrenzt. Als Trennzeichen wird üblicherweise ein Komma oder Semikolon verwendet. Falls das Trennzeichen zugleich Teil der Daten ist, werden die Daten in Anführungszeichen gesetzt. Ein Beispiel: Ein Datensatz besteht aus den Daten Name, Alter (in Jahren) und Größe (in Metern). Als Trennzeichen soll ein Komma verwendet werden. Dann würde der Inhalt der csv-Datei wie folgt aussehen:

```
Anton,24,"1,84"
Berta,22,"1,69"
```

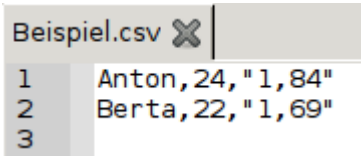
Man kann csv-Dateien wie Textdateien einlesen und speichern, muss sich dann aber selbst um die Trennzeichen und Anführungszeichen kümmern. Einfacher ist es, das `csv`-Paket zu nutzen. Dazu ergänzt man im Programmkopf die Zeile

```
package require csv
```

Beispiel: Zunächst wird mit einer Tabellenkalkulation eine csv-Datei erstellt. Die Tabelle hat drei Spalten: Name, Alter (in Jahren), Größe (in Metern):



Wenn man die csv-Datei mit einem Texteditor öffnet, sieht sie wie folgt aus:



Wie man sieht, wurde ein Komma als Trennzeichen verwendet, so dass die Größen-Angaben in Anführungszeichen gesetzt wurden. Das Programm zum Öffnen der Datei sieht wie folgt aus:

Listing 20.18: csv-Datei einlesen (Beispiel430.tcl)

```
1 #!/usr/bin/env tclsh
2
```

```
3  package require csv
4
5  set Dateiname Beispiel.csv
6
7  set f [open $Dateiname r]
8  while {[gets $f Zeile] >= 0} {
9          set Liste [::csv::split $Zeile ","]
10         puts $Liste
11 }
12 close $f
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_430.tcl
Anton 24 1,84
Berta 22 1,69
oliver@debian:~$
```

In Zeile 3 wird mit dem Befehl das `csv`-Paket eingebunden. In Zeile 8 wird jede Zeile aus der csv-Datei einzeln eingelesen. In Zeile 9 wird die eingelesene Zeile durch den Befehl `::csv::split $Zeile ","` gemäß den Regeln des csv-Formats in eine Liste umgewandelt. Als Trennzeichen wird ein Komma angegeben. Die Trennzeichen und Anführungszeichen werden entfernt, so dass man eine Liste mit den einzelnen Daten erhält. Bei csv-Dateien, die von Microsoft-Produkten erzeugt wurden, ist es meistens notwendig, den Befehl `::csv::split $Zeile ","` um die Option `-alternate` zu erweitern. Der Befehl sieht dann wie folgt aus: `::csv::split -alternate $Zeile ","`

Listing 20.19: csv-Datei erstellen (Beispiel431.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require csv
4
5  set Liste {Anton 24 1,84}
6
7  set Dateiname Ausgabe.csv
8  set f [open $Dateiname w]
9  puts $f [::csv::join $Liste ","]
10 close $f
```

Wenn man die Datei mit einem Texteditor öffnet, sieht sie wie folgt aus:

```
Ausgabe.csv
1   Anton,24,"1,84"
2
```

In Zeile 5 wird eine Liste mit drei Daten erzeugt. In Zeile 9 wird die Liste mit dem Befehl `::csv::join $Liste ","` gemäß den Regeln des csv-Formats zu einer Zeile zusammengefasst. Als Trennzeichen wird ein Komma verwendet.

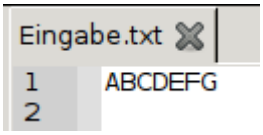
20.8 Binär-Dateien

Befehle:

- `set fDatei [open Eingabe.txt r]`
- `set fDatei [open Ausgabe.txt w]`
- `fconfigure $fDatei -translation binary`
- `seek $fDatei AnzahlStellen Bezugsposition`
- `set Zeichen [read $fDatei 1]`
- `puts -nonewline $fDatei $Zeichen`
- `[eof $fDatei]`
- `close $fDatei`

Tcl/Tk unterstellt standardmäßig, dass alle Dateien Text-Dateien sind. Man kann jedoch auch jede andere Datei Byte für Byte einlesen oder schreiben und spricht dann von Binär-Dateien. Der Befehl `fconfigure $fDatei -translation binary` legt fest, dass die Datei binär eingelesen bzw. gespeichert werden soll. Mit dem Befehl `seek` kann man die aktuelle Position in der Datei festlegen. Dabei wird die Position ausgehend von der Bezugsposition um eine bestimmte Anzahl Stellen verschoben. Wenn die Stellenanzahl größer als Null ist, wird der Zeiger nach rechts (also zum Dateiende hin) verschoben, ist die Anzahl kleiner als Null wird der Zeiger nach links (zum Dateianfang hin) verschoben. Als Bezugsposition gibt es `start` (= Dateianfang), `current` (= aktuelle Zeigerposition) und `end` (=Dateiende).

In dem folgenden Beispiel wird eine Textdatei verwendet, weil deren Bytes als Zeichen auf dem Bildschirm darstellbar sind. Sie können aber auch jede andere Datei (z. B. eine Bilddatei) verwenden. Erstellen Sie mit einem Texteditor eine Textdatei `Eingabe.txt` mit folgendem Inhalt:



Das Programm sieht wie folgt aus:

Listing 20.20: Binär-Datei einlesen bzw. erstellen (Beispiel321.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set fEingabe [open Eingabe.txt r]
4 fconfigure $fEingabe -translation binary
5
6 set fAusgabe [open Ausgabe.txt w]
7 fconfigure $fAusgabe -translation binary
8
9 while {1} {
```

```
10
11          set Zeichen [read $fEingabe 1]
12          if {[eof $fEingabe]} {
13                  close $fEingabe
14                  break
15          }
16
17          puts $Zeichen
18
19          puts -nonewline $fAusgabe $Zeichen
20 }
21
22 close $fAusgabe
23 puts "fertig"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_321.tcl
A
B
C
D
E
F
G

fertig
oliver@debian:~$
```

Das Programm hat die Eingabedatei `Eingabe.txt` Zeichen für Zeichen eingelesen und dann eine neue (binäre) Ausgabedatei `Ausgabe.txt` erstellt. In einem Hexadezimal-Betrachter (z. B. hexedit) sieht die Eingabedatei wie folgt aus:

```
oliver@debian: ~
00000000  41 42 43 44  45 46 47 0A                          ABCDEFG.
00000010
00000020
00000030
00000040
00000050
00000060
00000070
00000080
00000090
000000A0
000000B0
000000C0
000000D0
000000E0
000000F0
00000100
00000110
00000120
00000130
00000140
00000150
00000160
---  Eingabe.txt       --0x0/0x8-----------------------------------------------
```

Die Ausgabedatei sieht genauso aus:

```
oliver@debian: ~
00000000  41 42 43 44  45 46 47 0A                          ABCDEFG.
00000010
00000020
00000030
00000040
00000050
00000060
00000070
00000080
00000090
000000A0
000000B0
000000C0
000000D0
000000E0
000000F0
00000100
00000110
00000120
00000130
00000140
00000150
00000160
---  Ausgabe.txt       --0x0/0x8-----------------------------------------------
```

In Zeile 3 wird ein Filehandler `fEingabe` auf die Eingabedatei zum Lesen erzeugt. In Zeile 4 wird festgelegt, dass die Eingabedatei binär eingelesen werden soll. In Zeile 6 wird ein Filehandler `fAusgabe` auf die Ausgabedatei zum Schreiben erzeugt. In Zeile 7 wird festgelegt, dass die Ausgabedatei binär geschrieben werden soll.

Die `while`-Schleife in Zeile 9 liest jedes Zeichen (genauer: jedes Byte) der Eingabedatei

einzeln ein. Die Schleife läuft endlos. Der Abbruch ist in Zeile 14 definiert. In Zeile 11 wird mit dem Befehl `[read $fEingabe 1]` genau ein Zeichen eingelesen. Wenn man statt der 1 eine 3, also `[read $fEingabe 3]`, schreibt, werden drei Bytes eingelesen. Lässt man die Angabe weg, wird die Datei komplett eingelesen. Dann braucht man aber keine `while`-Schleife und sollte den möglichen Speicherverbrauch beachten.

Zeile 12 prüft, ob das Dateiende erreicht wurde. In diesem Fall wird die Eingabedatei geschlossen (Zeile 13) und die `while`-Schleife mit dem Befehl `break` beendet. In Zeile 17 wird das Zeichen in der Konsole ausgegeben. Beachten Sie, dass nicht jedes Byte einem darstellbaren Zeichen zugeordnet ist. In Zeile 19 wird das Zeichen, genauer gesagt das Byte, in die Ausgabedatei geschrieben. In Zeile 22 wird die Ausgabedatei geschlossen.

Listing 20.21: In einer Binärdatei springen (Beispiel384.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set fEingabe [open Eingabe.txt r]
4  fconfigure $fEingabe -translation binary
5
6  set Stelle 0
7  while {1} {
8    set Zeichen [read $fEingabe 1]
9    if {[eof $fEingabe]} {
10   break
11   }
12   puts "Stelle $Stelle: $Zeichen"
13   incr Stelle
14 }
15
16 seek $fEingabe 2 start
17 set Zeichen [read $fEingabe 1]
18 puts "2 Stellen nach dem Start: $Zeichen"
19 puts "Die aktuelle Stelle ist jetzt 3."
20
21 seek $fEingabe 3 current
22 set Zeichen [read $fEingabe 1]
23 puts "3 Stellen weiter als die aktuelle Stelle: $Zeichen"
24 puts "Die aktuelle Stelle ist jetzt 7."
25
26 seek $fEingabe -2 current
27 set Zeichen [read $fEingabe 1]
28 puts "2 Stellen vor der aktuellen Stelle: $Zeichen"
29 puts "Die aktuelle Stelle ist jetzt 6."
30
31 seek $fEingabe 0 current
32 set Zeichen [read $fEingabe 1]
33 puts "Aktuelle Stelle: $Zeichen"
34 puts "Die aktuelle Stelle ist jetzt 7."
35
36 seek $fEingabe -5 end
37 set Zeichen [read $fEingabe 1]
38 puts "5 Stellen vor dem Ende: $Zeichen"
39 puts "Die aktuelle Stelle ist jetzt 4."
40
```

```
41 close $fEingabe
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_384.tcl
Stelle 0: A
Stelle 1: B
Stelle 2: C
Stelle 3: D
Stelle 4: E
Stelle 5: F
Stelle 6: G
Stelle 7:

2 Stellen nach dem Start: C
Die aktuelle Stelle ist jetzt 3.
3 Stellen weiter als die aktuelle Stelle: G
Die aktuelle Stelle ist jetzt 7.
2 Stellen vor der aktuellen Stelle: F
Die aktuelle Stelle ist jetzt 6.
Aktuelle Stelle: G
Die aktuelle Stelle ist jetzt 7.
5 Stellen vor dem Ende: D
Die aktuelle Stelle ist jetzt 4.
oliver@debian:~$
```

In den Zeilen 6 bis 14 werden alle Zeichen der Textdatein eingelesen und angezeigt. In Zeile 16 springt man mit dem Befehl `seek $fDatei 2 start` an die zweite Stelle nach dem Start. In Zeile 17 wird ein Zeichen eingelesen. Dadurch verschiebt sich die aktuelle Lese-Position um eine Stelle weiter, so dass sie jetzt auf Stelle 3 steht. In Zeile 21 verschiebt man mit dem Befehl `seek $fEingabe 3 current` die aktuelle Lese-Position um drei Stellen nach rechts. In Zeile 26 verschiebt man mit dem Befehl `seek $fEingabe -2 current` die aktuelle Lese-Position um zwei Stellen nach links. In Zeile 36 setzt man mit dem Befehl `seek $fEingabe -5 end` die aktuelle Lese-Position fünf Stellen vor das Dateiende.

21 Dezimale, hexadezimale, binäre und ASCII-Darstellung

Befehle:

- `set Dezimal [scan $Zeichen %c]`
- `set Dezimal [scan $Hex %x]`
- `set Dezimal [scan $Bin %b]`
- `set Bin [format %08b $Dezimal]`
- `set Hex [format %02X $Dezimal]`
- `set Zeichen [format %c $Dezimal]`

Man kann Zahlen zwischen den Zahlensystemen Dezimal-, Binär und Hexadezimalsystem beliebig konvertieren. Ebenso zwischen der ASCII-Darstellung und dem ASCII-Code.

Listing 21.1: Zahlen und ASCII-Zeichen konvertieren (Beispiel322.tcl)

```
#!/usr/bin/env tclsh

set Zeichen "A"
set Dezimal [scan $Zeichen %c]
set Bin [format %08b $Dezimal]
set Hex [format %02x $Dezimal]
puts "Zeichen $Zeichen / dezimal $Dezimal / hexadezimal $Hex / binaer $Bin"

set Dezimal 66
set Bin [format %08b $Dezimal]
set Hex [format %02x $Dezimal]
set Zeichen [format %c $Dezimal]
puts "Zeichen $Zeichen / dezimal $Dezimal / hexadezimal $Hex / binaer $Bin"

#set Hex 0x43
set Hex 43
set Dezimal [scan $Hex %x]
set Bin [format %08b $Dezimal]
set Zeichen [format %c $Dezimal]
puts "Zeichen $Zeichen / dezimal $Dezimal / hexadezimal $Hex / binaer $Bin"

#set Bin "01000100"
#set Bin 0b01000100
set Bin 01000100
```

```
25 set Dezimal [scan $Bin %b]
26 set Hex [format %02x $Dezimal]
27 set Zeichen [format %c $Dezimal]
28 puts "Zeichen $Zeichen / dezimal $Dezimal / hexadezimal ↩
   $Hex / binaer $Bin"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_322.tcl
Zeichen A / dezimal 65 / hexadezimal 41 / binär 01000001
Zeichen B / dezimal 66 / hexadezimal 42 / binär 01000010
Zeichen C / dezimal 67 / hexadezimal 43 / binär 01000011
Zeichen D / dezimal 68 / hexadezimal 44 / binär 01000100
oliver@debian:~$
```

In den Zeilen 3 bis 7 wird mit einem Zeichen gestartet, dass in die verschiedenen Systeme konvertiert wird. In Zeile 4 wird das Zeichen (der Buchstabe) in den ASCII-Code umgewandelt, also in eine Dezimalzahl. In Zeile 5 wird die Dezimalzahl in eine 8-stellige Binärzahl umgewandelt. Dabei steht `08` für die Anzahl der Stellen und `b` für das Binärformat. Analog wird in Zeile 6 die Dezimalzahl in eine Hexadezimalzahl konvertiert. Dabei steht `02` für die Anzahl der Stellen und `x` für das Hexadezimalformat.
In den Zeilen 9 bis 13 wird mit einer Dezimalzahl gestartet.
In den Zeilen 15 bis 20 wird mit einer Hexadezimalzahl gestartet. Die Zeile 15 ist eine alternative Eingabemöglichkeit für eine hexadezimale Zahl, beginnend mit `0x`. In Zeile 17 wird die hexadezimale Zahl in eine Dezimalzahl umgewandelt.
In den Zeile 22 bis 28 wird mit einer Binärzahl gestartet. Die Zeilen 22 und 23 zeigen alternative Eingabemöglichkeiten für eine Binärzahl, entweder in Anführungszeichen oder mit einem vorangestellten `0b`.

22 Fehler während der Ausführung abfangen

Befehl:

- `catch {Anweisungen}`

Der Befehl `catch` fängt Fehler ab, die während der Programmausführung auftreten können. Zum Beispiel, wenn eine Datei geöffnet werden soll, die nicht existiert. Der Befehl gibt den Wert `0` zurück, wenn es keinen Fehler gab.

Listing 22.1: Fehlermeldung und Programmabbruch (Beispiel142.tcl)

```
1 #!/usr/bin/env tclsh
2
3 open Buch.txt
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_142.tcl
couldn't open "Buch.txt": no such file or directory
    while executing
"open Buch.txt"
    (file "./Beispiel_142.tcl" line 3)
oliver@debian:~$
```

Durch den Fehler in Zeile 3 bricht das Programm ab und Zeile 4 wird nicht mehr ausgeführt.

Listing 22.2: Fehlermeldung abfangen (Beispiel143.tcl)

```
1 #!/usr/bin/env tclsh
2
3 catch {open Buch.txt}
4 puts "Der Fehler wurde abgefangen und das Programm ↩
    fortgefuehrt."
```

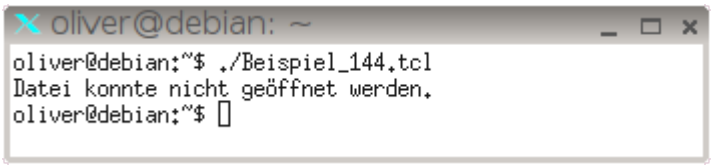
```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_143.tcl
Der Fehler wurde abgefangen und das Programm fortgeführt.
oliver@debian:~$
```

Der Fehler in Zeile 3 wird ignoriert und das Programm in Zeile 4 fortgeführt.

Listing 22.3: Fehlermeldung auswerten (Beispiel144.tcl)

```
1 #!/usr/bin/env tclsh
2
```

```
3 if {[catch {open Buch.txt}] == 0} {
4   puts "Datei konnte geoeffnet werden."
5 } else {
6   puts "Datei konnte nicht geoeffnet werden."
7 }
```



23 Mehrsprachigkeit

23.1 Mehrsprachigkeit

Ein Tcl/Tk-Programm kann sehr einfach mehrsprachig gemacht werden. Dazu legt man eine Sprachdatei an. Das ist eine Textdatei, die sowohl den Text gemäß Programmcode enthält und als auch dessen Übersetzung. Das Programm ersetzt während der Ausführung den Text aus dem Programmcode durch den Text, der in der Sprachdatei steht. Wenn man sowohl für Linux als auch für Windows Programme erstellt, sollte man idealerweise im Quellcode keine Umlaute verwenden und auch eine deutsche Sprachdatei erstellen. Erstellen Sie mit dem Editor eine Datei mit folgendem Inhalt:

Listing 23.1: Englische Sprachdatei (Beispiel145.tcl)

```
::msgcat::mcmset en {
"Hallo" "Hello"
"Wie geht es dir?" "How are you?"
}
```

```
en.msg
1  ::msgcat::mcmset en {
2  "Hallo" "Hello"
3  "Wie geht es dir?" "How are you?"
4  }
5
```

Beachten Sie das Sprachkürzel `en` in der ersten Zeile. Die Datei muss unter dem Dateinamen `en.msg` im Programmordner gespeichert werden.
Danach erstellen Sie eine französische Sprachdatei:

Listing 23.2: Französische Sprachdatei (Beispiel146.tcl)

```
::msgcat::mcmset fr {
"Hallo" "Allo"
"Wie geht es dir?" "Comment vas-tu?"
}
```

```
fr.msg
1  ::msgcat::mcmset fr {
2  "Hallo" "Allo"
3  "Wie geht es dir?" "Comment vas-tu?"
4  }
5
```

Beachten Sie das Sprachkürzel `fr` in der ersten Zeile. Die Datei muss unter dem Dateinamen `fr.msg` im Programmordner gespeichert werden.

Listing 23.3: Einen Text mehrsprachig ausgeben (Beispiel147.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require msgcat
4
5  ::msgcat::mclocale en
6  ::msgcat::mcload [file join [file dirname [info script]]]
7  puts [::msgcat::mc "Wie geht es dir?"]
8
9  ::msgcat::mclocale fr
10 ::msgcat::mcload [file join [file dirname [info script]]]
11 puts [::msgcat::mc "Wie geht es dir?"]
12
13 ::msgcat::mclocale de
14 puts [::msgcat::mc "Wie geht es dir?"]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_147.tcl
How are you?
Comment vas-tu?
Wie geht es dir?
oliver@debian:~$
```

In Zeile 3 wird das für die Mehrsprachigkeit notwendige Paket eingebunden. In Zeile 5 wird die Sprachdatei festgelegt, in diesem Fall die Datei `en.msg`. Dies muss erfolgen, bevor die Sprachdatei geladen wird. In Zeile 6 wird die Sprachdatei geladen. In Zeile 7 wird der deutsche Satz auf englisch ausgegeben. In den Zeilen 9 bis 11 erfolgt die Ausgabe in französisch. In den Zeilen 13 bis 14 erfolgt die Ausgabe in deutsch.
Möchte man einen Text, der Variablen enthält, mehrsprachig ausgeben, muss man im Text Platzhalter (siehe Kapitel 12.8 auf Seite 147) für die Variablen verwenden. Die Sprachdatei sieht dann z. B. wie folgt aus:

Listing 23.4: Sprachdatei mit Platzhalter (Beispiel148.tcl)

```
1 ::msgcat::mcmset en {
2 "Die Zahl ist %d" "The number is %d"
3 }
```

```
en.msg
1  ::msgcat::mcmset en {
2  "Die Zahl ist %d" "The number is %d"
3  }
4
```

Listing 23.5: Einen Text mit Variablen mehrsprachig ausgeben (Beispiel149.tcl)

```
#!/usr/bin/env tclsh

package require msgcat
::msgcat::mclocale en
::msgcat::mcload [file join [file dirname [info script]]]

set Zahl 123
puts [::msgcat::mc "Die Zahl ist %d" $Zahl]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_149.tcl
The number is 123
oliver@debian:~$
```

Wenn man in dem mehrsprachigen Text den Inhalt von Variablen ausgeben will, muss man Platzhalter verwenden. Der Platzhalter muss auch in der Sprachdatei enthalten sein.

23.2 Betriebssystemunabhängige Textausgabe

Tcl/Tk-Programme können generell auf den Betriebssystemen Windows, OS X und Linux ausgeführt werden. Allerdings verwenden die Betriebssysteme unterschiedliche Zeichencodes für deutsche Sonderzeichen wie z. B. die Umlaute ä, ö und ü. Damit es keine Probleme bei der Textausgabe gibt, sollte man folgendes beachten:

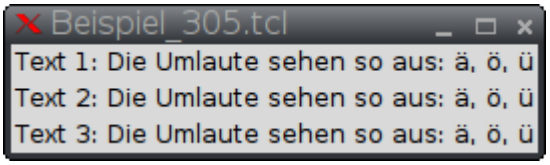
- im Quellcode keine Umlaute verwenden
- auch für die deutsche Sprache eine Sprachdatei erstellen

Die deutsche Sprachdatei hat z. B. folgenden Inhalt:

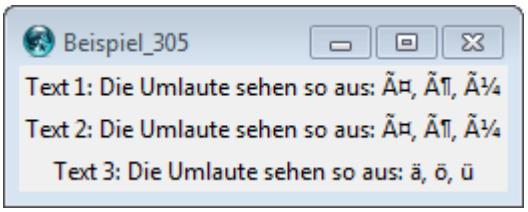
```
de.msg
::msgcat::mcmset de {
"Text 2: Die Umlaute sehen so aus: ä, ö, ü" "Text 2: Die Umlaute sehen so aus: ä, ö, ü"
"Text 3: Die Umlaute sehen so aus: ae, oe, ue" "Text 3: Die Umlaute sehen so aus: ä, ö, ü"
}
```

```
#!/usr/bin/env wish

package require msgcat
::msgcat::mclocale de
::msgcat::mcload [file join [file dirname [info script]]]

label .lbText1 -text "Text 1: Die Umlaute sehen so aus: ä, ö, ü"
label .lbText2 -text [::msgcat::mc "Text 2: Die Umlaute sehen so aus: ä, ö, ü"]
label .lbText3 -text [::msgcat::mc "Text 3: Die Umlaute sehen so aus: ae, oe, ue"]

pack .lbText1
pack .lbText2
pack .lbText3
```

Wenn das Programm unter Linux erstellt und auch ausgeführt wurde, sieht das Ergebnis wie folgt aus:



Die Umlaute werden alle korrekt dargestellt. Wenn man dasselbe Programm unter Windows ausführt, sieht es jedoch so aus:



Wie man sieht wird nur der Text 3 korrekt dargestellt, weil auf Umlaute im Quellcode verzichtet wurde und die deutsche Sprachdatei benutzt wird. Bei Text 2 wird zwar ebenfalls die deutsche Sprachdatei benutzt, aber im Quellcode wurden Umlaute verwendet. Diese werden unter Windows bei der Ausführung des Quellcodes anderes interpretiert als die Umlaute in der Sprachdatei.

24 Programm eine Zeit lang anhalten

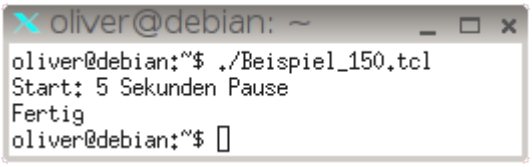
Befehl:

- `after Millisekunden`

Der Befehl `after` hält das Programm für eine bestimmte Anzahl an Millisekunden an.

Listing 24.1: Programm anhalten (Beispiel150.tcl)

```
#!/usr/bin/env tclsh
puts "Start: 5 Sekunden Pause"
after 5000
puts "Fertig"
```



25 Andere Programme starten

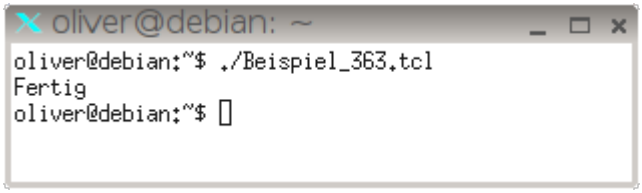
Befehl:

- `exec`

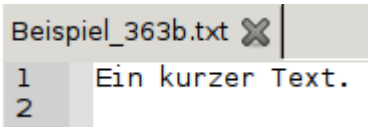
Es kann sehr nützlich sein, aus dem eigenen Programm heraus andere, bereits vorhandene Programme zu nutzen. Wenn man beispielsweise ein Bildbetrachtungs-Programm geschrieben hat und dem Anwender die Möglichkeit zur Bildbearbeitung bieten will, kann man (statt selbst Bildbearbeitungsfunktionen zu programmieren) ein Bildbearbeitungsprogramm aufrufen und die Bilddatei übergeben. Dazu verwendet man den Befehl `exec`.

Listing 25.1: Anderes Programm starten und auf dessen Beendigung warten (Beispiel363.tcl)

```
1 #!/usr/bin/env tclsh
2
3 exec geany Beispiel_363b.txt
4 puts "Fertig"
```



In Geany:



In Zeile 3 wird das Programm `geany` aufgerufen und die Datei `Beispiel_363b.txt` übergeben. Das Tcl/Tk-Programm wartet, bis das Programm `geany` beendet wird. Erst danach wird das Tcl/Tk-Programm weiter ausgeführt.

Listing 25.2: Anderes Programm starten und Tcl/Tk-Programm weiter ausführen (Beispiel366.tcl)

```
1 #!/usr/bin/env tclsh
2
3 exec geany Beispiel_363b.txt &
4 puts "Fertig"
```

```
test@debian: ~
test@debian:~$ ./Beispiel366.tcl
Fertig
test@debian:~$
```

In Geany:

```
Beispiel_363b.txt
1   Ein kurzer Text.
2
```

In Zeile 3 wurde ein `&`-Zeichen an das Zeilenende hinzugefügt. Dadurch legt man fest, dass das Tcl/Tk-Programm sofort weiter ausgeführt und nicht auf die Beendigung des anderen Programms wartet.

Listing 25.3: Liste expandieren bei Übergabe an andere Programme (Beispiel378.tcl)

```
1 #!/usr/bin/env tclsh
2
3 set Liste {Datei1 Datei2 Datei3}
4 exec zip Paket.zip {*}$Liste
```

In Zeile 3 wird eine Liste mit drei Dateinamen erstellt. In Zeile 4 startet der `exec`-Befehl das Programm `zip`. An das `zip`-Programm werden der Name der gezippten Datei `Paket.zip` und durch den Befehl `{*}$Liste` außerdem die drei Dateinamen `Datei1 Datei2 Datei3` übergeben, damit das `zip`-Programm die drei Dateien zusammenpackt.

26 Tcl mit C/C++ kombinieren

Es gibt verschiedene Möglichkeiten Tcl mit C/C++ zu kombinieren. Man kann sowohl ein Tcl-Programm mit C/C++-Code anreichern als auch umgekehrt in einem C/C++-Programm Tcl-Code ausführen. Allerdings setzen beide Varianten tiefer gehendes Wissen voraus.
Aus meiner Sicht ist es am einfachsten, wenn man zunächst das Tcl/Tk-Programm erstellt und misst, welche Stellen zu langsam sind. Dann schreibt man genau diese Funktionalität als C/C++-Programm. Anschließend ersetzt man in seinem Tcl/Tk-Programm den Code durch den Aufruf des C/C++-Programms. Das geschieht mit dem `exec`-Befehl (siehe Kapitel 25 auf Seite 273). Dabei kann man auch Werte übergeben.
Das folgende Beispiel berechnet eine Summe, in dem immer wieder 0,1 addiert wird. Das C++-Programm sieht wie folgt aus:

Listing 26.1: C++-Programm (Beispiel559main.cpp)

```
1  #include <iostream>
2  #include <time.h>
3  #include <stdlib.h>
4
5  using namespace std;
6
7  int main(int argc, char* argv[])
8  {
9      double time=0.0;
10     double tstart;
11     double summe;
12     unsigned long i;
13     unsigned long anzahl ;
14
15     anzahl = strtoul(argv[1], NULL, 10);
16     summe =0;
17     tstart = clock();
18     for(i=0;i<=anzahl;++i) {
19         summe=summe+0.1;
20     }
21     time += clock() - tstart;
22     time = time/CLOCKS_PER_SEC;
23     cout.precision(17);
24     cout << "Summe=" << summe << endl;
25     cout << "Zeit in C++=" << time << " Sekunden" << endl;
26     return 0;
27 }
```

Das Programm wird kompiliert und die ausführbare Datei als `Summe` gespeichert. Das C++-Programm nimmt in Zeile 7 die Argumente aus Tcl entgegen. Die Rückgabe der Werte an das aufrufende Tcl-Programm erfolgt in den Zeilen 24 und 25 mit dem Befehl `cout`.

Das Tcl/Tk-Programm sieht wie folgt aus:

Listing 26.2: Tcl/Tk-Programm (Beispiel559.tcl)

```
1  #!/usr/bin/env tclsh
2  set Anzahl 1234567
3
4  puts "Reines Tcl:"
5  set Summe 0
6  set Start [clock clicks -milliseconds]
7  for {set i 0} {$i <= $Anzahl} {incr i} {
8          set Summe [expr double($Summe) + 0.1]
9  }
10 set Stop [clock clicks -milliseconds]
11 set Zeit [expr (double($Stop) - double($Start)) / 1000.0]
12 puts "Summe=$Summe"
13 puts "Zeit=$Zeit Sekunden"
14
15 puts "---------"
16
17 puts "Tcl mit C++:"
18 set Start [clock clicks -milliseconds]
19 set Text [exec /home/oliver/Summe $Anzahl]
20 puts $Text
21 set Stop [clock clicks -milliseconds]
22 set Zeit [expr (double($Stop) - double($Start)) / 1000.0]
23 puts "Zeit Tcl mit C++=$Zeit Sekunden"
```

```
oliver@ubuntu: ~
oliver@ubuntu:~$ ./Beispiel559.tcl
Reines Tcl:
Summe=123456.80000269825
Zeit=7.371 Sekunden
---------
Tcl mit C++:
Summe=123456.80000269825
Zeit in C++=0.0055510000000000004 Sekunden
Zeit Tcl mit C++=0.008 Sekunden
oliver@ubuntu:~$
```

In Zeile 2 wird festgelegt, wie oft die Zahl 0,1 addiert werden soll. In den Zeilen 4 bis 13 wird die Summe nur mit Tcl-Befehlen berechnet. In den Zeilen 17 bis 23 wird für die Berechnung der Summe das C++-Programm verwendet. In Zeile 19 wird das C++-Programm aufgerufen. Dabei wird die Anzahl der Wiederholungen als Argument übergeben. Die Rückgabe aus dem C++-Programm (mittels der Funktion `cout`) wird in die Variable `Text` gespeichert.
Wie man sieht, dauert die Berechnung der Summe mit reinem Tcl-Code rund 7 Sekunden, mit dem C++-Programm aber nur 0,008 Sekunden. Die Berechnung ist somit rund 1.000 mal schneller.
Auch bei Programmen mit grafischer Benutzeroberfläche funktioniert dieses Vorgehen. Das C++-Programm sieht wie folgt aus:

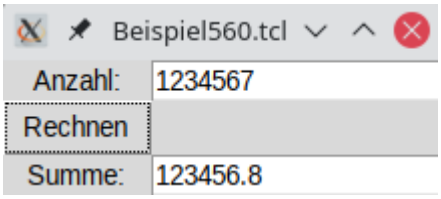
Listing 26.3: C++-Programm (Beispiel560main.cpp)

```
#include <iostream>
#include <stdlib.h>

using namespace std;

int main(int argc, char* argv[])
{
    double summe;
    unsigned long  i;
    unsigned long anzahl ;

    anzahl = strtoul(argv[1], NULL, 10);
    summe =0;
    for(i=0;i<=anzahl;++i) {
        summe=summe+0.1;
    }
    cout.precision(17);
    cout << summe << endl;
    return 0;
}
```

Das Programm wird kompiliert und die ausführbare Datei als `Summe` gespeichert.
Das Tcl/Tk-Programm sieht wie folgt aus:

Listing 26.4: Tcl/Tk-Programm mit Gui (Beispiel560.tcl)

```
#!/usr/bin/env wish

set Summe 0
ttk::frame .fr
ttk::label .fr.lbAnzahl -text "Anzahl:"
ttk::entry .fr.enAnzahl -textvariable Anzahl
ttk::button .fr.btRechnen -text Rechnen -command {set Summe [exec /home/oliver/Summe $Anzahl]}
ttk::label .fr.lbSumme -text "Summe:"
ttk::entry .fr.enSumme -textvariable Summe

grid .fr.lbAnzahl -row 0 -column 0
grid .fr.enAnzahl -row 0 -column 1
grid .fr.btRechnen -row 1 -column 0
grid .fr.lbSumme -row 2 -column 0
grid .fr.enSumme -row 2 -column 1

pack .fr
```



27 Eigene Befehle erzeugen

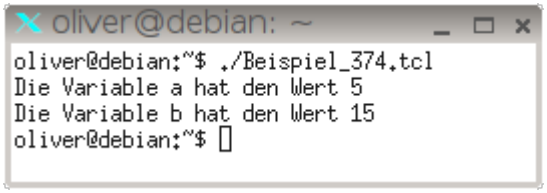
Befehl:

- `eval`

Man kann während der Programmausführung eigene Befehle erzeugen. Zunächst bildet man den Befehl und speichert ihn als Text ab. Danach übergibt man den Text an den `eval`-Befehl. Dieser führt dann den Befehl aus.

Listing 27.1: Eigene Befehle erzeugen (Beispiel374.tcl)

```
1  #!/usr/bin/env tclsh
2
3  set Befehl1 {set a 5}
4  set Befehl2 {puts "Die Variable a hat den Wert $a"}
5  set Befehl3 {set b [expr $a + 10]}
6  set Befehl4 {puts "Die Variable b hat den Wert $b"}
7
8  eval $Befehl1
9  eval $Befehl2
10 eval $Befehl3
11 eval $Befehl4
```



In den Zeilen 3 bis 6 werden verschiedene Befehle erzeugt und in Variablen gespeichert. In den Zeilen 8 bis 11 führt der `eval`-Befehl die Befehle aus.

28 Formel-Eingabe durch den Anwender

Man braucht nicht immer den `eval`-Befehl, um während der Programmausführung Befehle auszuführen. Im folgenden Beispiel gibt der Anwender eine Formel ein, die anschließend vom Programm zur Berechnung eines Ergebnisses verwendet wird.

Listing 28.1: Eingabe einer Formel durch den Anwender (Beispiel151.tcl)

```
1  #!/usr/bin/env tclsh
2
3  puts "Geben Sie eine Formel mit den Variablen x und y ein,
       z. B. x-y+4.0:"
4  set Eingabe [gets stdin]
5
6  for {set x 8} {$x <= 10} {incr x} {
7    for {set y 2} {$y <= 3} {incr y} {
8      set Formel $Eingabe
9      set Formel [string map "x double(x)" $Formel]
10     set Formel [string map "y double(y)" $Formel]
11     set Formel [string map "x $x" $Formel]
12     set Formel [string map "y $y" $Formel]
13     if {[catch {set Ergebnis [expr $Formel]}] != 0} {
14       set Ergebnis ""
15     }
16     puts "x=$x y=$y Formel:$Formel Ergebnis:$Ergebnis"
17   }
18 }
```

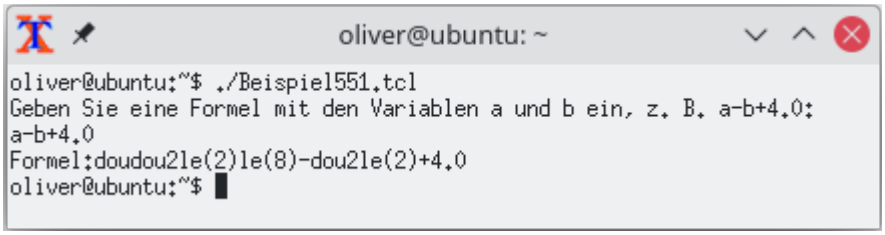
```
oliver@ubuntu: ~
oliver@ubuntu:~$ ./Beispiel151.tcl
Geben Sie eine Formel mit den Variablen x und y ein, z. B. x-y+4.0:
x-y+4.0
x=8 y=2 Formel:double(8)-double(2)+4.0 Ergebnis:10.0
x=8 y=3 Formel:double(8)-double(3)+4.0 Ergebnis:9.0
x=9 y=2 Formel:double(9)-double(2)+4.0 Ergebnis:11.0
x=9 y=3 Formel:double(9)-double(3)+4.0 Ergebnis:10.0
x=10 y=2 Formel:double(10)-double(2)+4.0 Ergebnis:12.0
x=10 y=3 Formel:double(10)-double(3)+4.0 Ergebnis:11.0
oliver@ubuntu:~$
```

In Zeile 4 wird eine Formel eingegeben und gespeichert. In Zeile 8 wird die Eingabe in die Variable `Formel` übertragen. In den Zeilen 9 und 10 werden die Buchstaben `x` und `y` durch `double(x)` bzw. `double(y)` ersetzt. In Zeile 11 wird der Buchstabe `x` durch `$x` ersetzt, in Zeile 12 der Buchstabe `y` durch `$y`. In Zeile 13 wird das Ergebnis gemäß der Formel berechnet. Es ist ratsam, mögliche Fehler, die bei der Berechnung des Ergebnisses auftreten können (z. B. Division durch Null oder unsinnige Formeln) mit dem `catch`-Befehl abzufangen.

Bei der Wahl der Buchstaben für die Formel muss man aufpassen, dass man keine Buchstaben verwendet, die eventuell mehrfach ersetzt werden.

Listing 28.2: Mehrfache Ersetzung in der Formel (Beispiel551.tcl)

```
1  #!/usr/bin/env tclsh
2
3  puts "Geben Sie eine Formel mit den Variablen a und b ein,
       z. B. a-b+4.0:"
4  set Eingabe [gets stdin]
5
6  set a 8
7  set b 2
8  set Formel $Eingabe
9  set Formel [string map "a double(a)" $Formel]
10 set Formel [string map "b double(b)" $Formel]
11 set Formel [string map "a $a" $Formel]
12 set Formel [string map "b $b" $Formel]
13 puts "Formel:$Formel"
```



In diesem Beispiel wurden für die Formel die Buchstaben `a` und `b` gewählt. Die Ersetzung in Zeile 9 führt zu der Formel `double(a)-b+4.0`. Bei der Ersetzung in Zeile 10 wird jetzt aber der Buchstabe `b` auch in `double(a)` ersetzt, so dass die Formel zu `doudouble(b)le(a)-double(b)+4.0` wird. In Zeile 11 werden dann die Buchstaben `a` und `b` durch die Zahlen `8` und `2` ersetzt. Wie man sieht, ist eine falsche Formel entstanden.

29 Befehle aus einer Textdatei ausführen

Befehl:

- `source`

Wenn man eine Textdatei mit Befehlen erstellt, kann man mit dem Befehl `source` die Befehle aus dieser Textdatei ausführen. Dadurch hat man die Möglichkeit, seinen Quellcode auf verschiedene Dateien zu verteilen und diese im Hauptprogramm aufzurufen.
Erstellen Sie mit einem Editor die Textdatei `Beispiel_375.txt` mit folgendem Inhalt:

Beispiel_375.txt

```
1  set a 5
2  puts "Die Variable a hat den Wert $a"
3  set b [expr $a + 10]
4  puts "Die Variable b hat den Wert $b"
```

Das Programm hat folgenden Code:

Listing 29.1: Hauptprogramm (Beispiel375.tcl)

```
1 #!/usr/bin/env tclsh
2
3 source Beispiel_375.txt
```

oliver@debian: ~

```
oliver@debian:~$ ./Beispiel_375.tcl
Die Variable a hat den Wert 5
Die Variable b hat den Wert 15
oliver@debian:~$
```

30 Namensräume

Befehl 1: Namensraum mit einer Prozedur erzeugen

```
namespace eval Namensraum {
 namespace export Prozedur
 variable Variable Wert
 proc Prozedur {} {
 variable Variable
 Anweisungen
 }
}
import ::Namensraum::Prozedure
```

Befehl 2: Leeren Namensraum erzeugen und anschließend eine Prozedur zuordnen

```
namespace eval Namensraum {
 namespace export Prozedur
 variable Variable Wert
}
proc ::Namensraum::Prozedur {} {
 variable Variable
 Anweisungen
}
import ::Namensraum::Prozedure
```

Wenn man Prozeduren in verschiedenen Programmen verwenden möchte, besteht die Gefahr, dass es zu Namenskonflikten kommt. Deshalb speichert man die wiederverwendbaren Prozeduren in eigenen Namensräumen (namespaces).

- Der globale Namensraum hat als Identifizierung zwei Doppelpunkte `::`
- Ein Namensraum (innerhalb des globalen Namensraums) hat den Namen `::Namensraum`
- Eine Prozedur innerhalb des Namensraums hat den Namen `::Namensraum::Prozedur`

Der Befehl `namespace eval ::Namensraum {}` erzeugt einen leeren Namensraum.
Der Befehl `namespace export Prozedur` legt fest, welche Prozeduren von außerhalb des Namensraums aufgerufen werden dürfen.
Außerhalb des Namensraum importiert man eine Prozedur aus einem Namensraum mit dem Befehl `namespace import ::Namensraum::Prozedur`. Beim Importieren kann man auch mit der Wildcard `*` arbeiten, z.B. importiert `namespace import ::Namensraum::*` alle Prozeduren des Namensraums.
Innerhalb des Namensraums definiert man mit dem Befehl `variable Variable Wert` alle Variablen, die innerhalb des Namensraums global gültig sein sollen. Innerhalb einer Prozedur kann man auf diese Variable zugreifen, wenn man zu Beginn der Prozedur ebenfalls den Befehl `variable Variable` einfügt (dies entspricht dem Befehl `global`). Alle

anderen Variablen, die innerhalb einer Prozedur definiert werden, sind nur innerhalb der Prozedur gültig.

Listing 30.1: Namensraum mit einer Prozedur erzeugen (Beispiel152.tcl)

```
1  #!/usr/bin/env tclsh
2
3  namespace eval Mathematik {
4    namespace export Plus5
5    proc Plus5 {Zahl} {
6      return [incr Zahl 5]
7    }
8  }
9
10 namespace import ::Mathematik::Plus5
11 puts "3 + 5 = [::Mathematik::Plus5 3]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_152.tcl
3 + 5 = 8
oliver@debian:~$
```

Listing 30.2: Leeren Namensraum erzeugen und danach eine Prozedur in den Namensraum hinzufügen (Beispiel153.tcl)

```
1  #!/usr/bin/env tclsh
2
3  namespace eval Mathematik {
4    namespace export Plus5
5  }
6
7  proc ::Mathematik::Plus5 {Zahl} {
8    return [incr Zahl 5]
9  }
10
11 namespace import ::Mathematik::Plus5
12 puts "3 + 5 = [::Mathematik::Plus5 3]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_153.tcl
3 + 5 = 8
oliver@debian:~$
```

In den Zeilen 3 bis 5 wird ein leerer Namensraum erzeugt. Dabei muss man schon festlegen, welche Prozeduren außerhalb des Namensraums aufrufbar sein sollen. In den Zeilen 7 bis 9 wird die Prozedur `Plus5` dem Namensraum hinzugefügt.

Listing 30.3: Mehrere Prozeduren importieren (Beispiel154.tcl)

```
1  #!/usr/bin/env tclsh
```

```
2
3 namespace eval Mathematik {
4   namespace export Plus5 Plus10
5   proc Plus5 {Zahl} {
6     return [incr Zahl 5]
7   }
8
9   proc Plus10 {Zahl} {
10    return [incr Zahl 10]
11  }
12 }
13
14 namespace import ::Mathematik::Plus*
15 puts "3 + 5 = [::Mathematik::Plus5 3]"
16 puts "3 + 10 = [::Mathematik::Plus10 3]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_154.tcl
3 + 5 = 8
3 + 10 = 13
oliver@debian:~$
```

In Zeile 4 werden mehrere Prozeduren exportiert. In Zeile 14 werden aus dem Namensraum `Mathematik` alle Prozeduren importiert, deren Namen mit Plus beginnen.

Listing 30.4: Innerhalb des Namensraums global gültige Variable (Beispiel155.tcl)

```
1 #!/usr/bin/env tclsh
2
3 namespace eval Mathematik {
4   namespace export Plus5
5   variable Wert 5
6   proc Plus5 {Zahl} {
7     variable Wert
8     return [incr Zahl $Wert]
9   }
10 }
11
12 namespace import ::Mathematik::Plus5
13 puts "3 + 5 = [::Mathematik::Plus5 3]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_155.tcl
3 + 5 = 8
oliver@debian:~$
```

In Zeile 4 wird die Variable `Wert` definiert und ihr die Zahl `5` zugewiesen. Diese Variable ist im gesamten Namensraum gültig. In Zeile 7 wird durch den Befehl `variable Wert` auf die (im Namensraum globale) Variable `Wert` zugegriffen. Dies entspricht dem Befehl `global`.

Listing 30.5: Innerhalb des Namensraums global gültige Variable behalten ihren Wert (Beispiel156.tcl)

```
1  #!/usr/bin/env tclsh
2
3  namespace eval Beispiel {
4    namespace export Zahl
5    variable Zahl 0
6
7    proc Zahl {} {
8      variable Zahl
9      incr Zahl
10     return $Zahl
11   }
12 }
13
14 namespace import ::Beispiel::Zahl
15 puts "1. Aufruf: [::Beispiel::Zahl]"
16 puts "2. Aufruf: [::Beispiel::Zahl]"
17 puts "3. Aufruf: [::Beispiel::Zahl]"
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_156.tcl
1. Aufruf: 1
2. Aufruf: 2
3. Aufruf: 3
oliver@debian:~$
```

In Zeile 5 wird die Variable `Zahl` global innerhalb des Namensraums definiert und ihr der Wert `0` zugewiesen. Die Prozedur `Zahl` (Zeilen 7 bis 11) erhöht die Variable `Zahl` um 1. Jeder Aufruf der Prozedur aus dem globalen Namensraum heraus (Zeilen 15 bis 17) erhöht die Variable `Zahl` um 1. Der Wert der Variable `Zahl` innerhalb des Namensraums `Beispiel` bleibt erhalten.

31 Datenbank sqlite3

In Tcl/Tk kann man sehr bequem die SQL-Datenbank sqlite3 verwenden. Sqlite3 speichert die gesamte Datenbank normalerweise in einer einzigen Datei. Man kann die Datenbank aber auch ausschließlich im Arbeitsspeicher halten.

Tabelle 31.1: Einige SQL-Befehle

SQL-Befehl	Beschreibung

`create table Tabelle ( \`
`  ID integer primary key autoincrement, \`
`  Ganzzahl integer, \`
`  Fliesskommazahl real, \`
`  Text text)`
	Tabelle erzeugen

`drop table if exists Tabelle`	Tabelle löschen
`create index IndexName on` `  Tabelle(Spalte,Spalte,...)`	Index anlegen
`insert into Tabelle` `  values (Wert,Wert,...)`	Datensatz (Zeile) hinzufügen
`update Tabelle set` `  Spalte=NeuerWert where ID=3`	Datensatz ändern
`delete from Tabelle where ID=3`	Datensatz löschen
`select count(*) from Tabelle`	Anzahl Datensätze
`select * from Tabelle where ID=3`	Kompletten Datensatz abfragen
`select Spalte1, Spalte2 from` `  Tabelle where ID=3`	Zwei Spalten eines bestimten Datensatzes abfragen
`select distinct Spalte from Tabelle`	Spalte abfragen, ohne Dubletten

`select * from Tabelle1 left outer`
`  join Tabelle2 on`
`  Tabelle1.ID=Tabelle2.ID`
`  where Tabelle1.ID=3`
	Tabellen verknüpfen

`select * from Tabelle` `  order by Spalte`	Tabelle aufsteigend sortieren

Tabelle 31.1: Einige SQL-Befehle

SQL-Befehl	Beschreibung
`select * from Tabelle order by Spalte desc`	Tabelle absteigend sortieren
`select * from Tabelle order by random()`	Tabelle zufällig sortieren
`select Spalte1 * (Spalte2 + Spalte3) from Tabelle`	Rechnen
`select sum(Spalte) from Tabelle`	Summieren
`select sum(Spalte1) from Tabelle group by Spalte2`	Gruppieren

Listing 31.1: sqlite3 (Beispiel157.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require sqlite3
4
5  set Datenbank [file join [file dirname [info script]] ↩
     MeineDB] ; # entfaellt, wenn die Datenbank im ↩
     Arbeitsspeicher gehalten wird
6
7  proc DatenbankErstellen {} {
8    global db
9
10   db eval {create table Person( \
11     ID integer primary key autoincrement, \
12     Name text, \
13     AlterJahre integer, \
14     IDAdresse integer
15   )}
16
17   db eval {create table Adresse( \
18     ID integer primary key autoincrement, \
19     Strasse text, \
20     PLZ text, \
21     Ort text
22   )}
23 }
24
25 proc DatenbankOeffnen {} {
26   global Datenbank
27   global db
28   if {[file exist $Datenbank] == 0} {
29     # Datenbank existiert nicht
30     sqlite3 db $Datenbank ; # Alternativ: sqlite3 db "↩
         :memory:"
```

```
31      DatenbankErstellen
32    } else {
33      # Datenbank existiert
34      sqlite3 db $Datenbank
35    }
36  }
37
38  proc DatenbankSchliessen {} {
39    global db
40    catch db close
41  }
42
43  proc DatenbankLoeschen {} {
44    global db
45    db eval {drop table Person}
46    db eval {drop table Adresse}
47  }
48
49  proc PersonEinfuegen {Name Alter Adresse} {
50    global db
51    db eval {insert into Person values(
        NULL,$Name,$Alter,$Adresse)}
52  }
53
54  proc AdresseEinfuegen {Strasse PLZ Ort} {
55    global db
56    db eval {insert into Adresse values(
        NULL,$Strasse,$PLZ,$Ort)}
57  }
58
59  proc PersonAnzeigen {ID} {
60    global db
61    puts [db eval {select
        Person.Name,Person.AlterJahre,Adresse.Strasse,Adresse.PLZ,Adresse.Or
         from Person left outer join Adresse ON
        Person.IDAdresse=Adresse.ID where Person.ID=$ID}]
62  }
63
64  proc PersonenAnzeigenUndZaehlen {} {
65    global db
66    puts "\n"
67    puts "Prozedur PersonenAnzeigenUndZaehlen:"
68    set AnzahlPersonenBis39 0
69    set AnzahlPersonenAb40 0
70    db eval {select
        Person.Name,Person.AlterJahre,Adresse.Strasse,Adresse.PLZ,Adresse.Or
         from Person left outer join Adresse ON
        Person.IDAdresse=Adresse.ID} Ergebnis {
71      parray Ergebnis
72      if {$Ergebnis(AlterJahre)<40} {
73        incr AnzahlPersonenBis39
74      } else {
75        incr AnzahlPersonenAb40
```

```
76      }
77    }
78    puts "Personen bis 39 Jahre: $AnzahlPersonenBis39"
79    puts "Personen ab 40 Jahre: $AnzahlPersonenAb40"
80    puts "\n"
81  }
82
83  proc EinfuegenOhneTransaction {} {
84    global db
85    for {set i 0} {$i <=300} {incr i} {
86      AdresseEinfuegen "Neue Str. 9" 65432 Musterstadt
87    }
88  }
89
90  proc EinfuegenMitTransaction {} {
91    global db
92    db eval {begin transaction}
93    for {set i 0} {$i <=300} {incr i} {
94      AdresseEinfuegen "Neue Str. 9" 65432 Musterstadt
95    }
96    db eval {end transaction}
97  }
98
99  # Hauptprogramm
100
101 DatenbankOeffnen
102
103 db eval {begin transaction}
104 AdresseEinfuegen "Hauptstr. 5" 65432 Musterstadt
105 AdresseEinfuegen "Nebenstr. 7" 23456 Neustadt
106 db eval {end transaction}
107
108 db eval {begin transaction}
109 PersonEinfuegen Anton 38 1
110 PersonEinfuegen Berta 36 1
111 PersonEinfuegen Caesar 41 2
112 db eval {end transaction}
113
114 PersonAnzeigen 1
115 PersonAnzeigen 2
116 PersonAnzeigen 3
117
118 PersonenAnzeigenUndZaehlen
119
120 puts "Einfuegen ohne Transaktion:"
121 set Start [clock clicks -milliseconds]
122 EinfuegenOhneTransaction
123 set Stopp [clock clicks -milliseconds]
124 set Dauer [expr ($Stopp - $Start)/1000.0]
125 puts "Dauer: $Dauer Sekunden"
126
127 puts "Einfuegen mit Transaktion:"
128 set Start [clock clicks -milliseconds]
```

```
129 EinfuegenMitTransaction
130 set Stopp [clock clicks -milliseconds]
131 set Dauer [expr ($Stopp - $Start)/1000.0]
132 puts "Dauer: $Dauer Sekunden"
133
134 DatenbankSchliessen
135
136 file delete -force $Datenbank
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_157.tcl
Anton 38 {Hauptstr. 5} 65432 Musterstadt
Berta 36 {Hauptstr. 5} 65432 Musterstadt
Cäsar 41 {Nebenstr. 7} 23456 Neustadt


Prozedur PersonenAnzeigenUndZaehlen:
Ergebnis(*)          = Name AlterJahre Strasse PLZ Ort
Ergebnis(AlterJahre) = 38
Ergebnis(Name)       = Anton
Ergebnis(Ort)        = Musterstadt
Ergebnis(PLZ)        = 65432
Ergebnis(Strasse)    = Hauptstr. 5
Ergebnis(*)          = Name AlterJahre Strasse PLZ Ort
Ergebnis(AlterJahre) = 36
Ergebnis(Name)       = Berta
Ergebnis(Ort)        = Musterstadt
Ergebnis(PLZ)        = 65432
Ergebnis(Strasse)    = Hauptstr. 5
Ergebnis(*)          = Name AlterJahre Strasse PLZ Ort
Ergebnis(AlterJahre) = 41
Ergebnis(Name)       = Cäsar
Ergebnis(Ort)        = Neustadt
Ergebnis(PLZ)        = 23456
Ergebnis(Strasse)    = Nebenstr. 7
Personen bis 39 Jahre: 2
Personen ab 40 Jahre: 1


Einfügen ohne Transaktion:
Dauer: 15.501 Sekunden
Einfügen mit Transaktion:
Dauer: 0.077 Sekunden
oliver@debian:~$
```

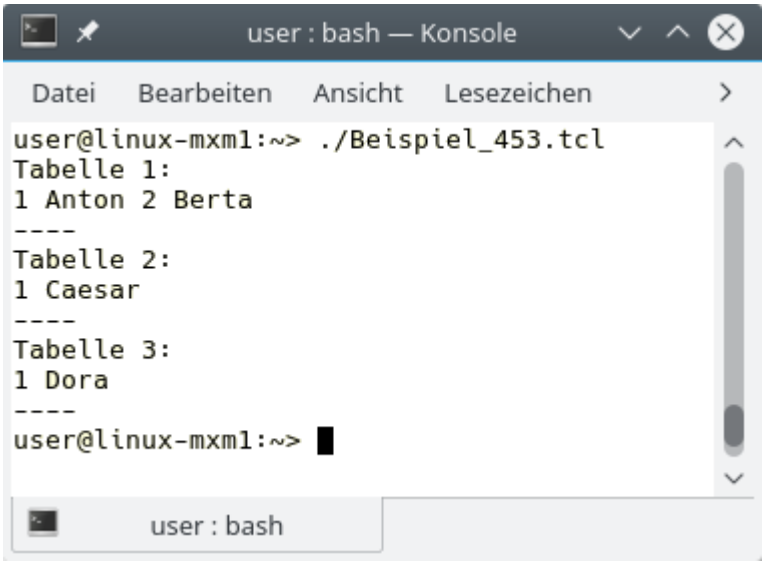
In Zeile 3 wird das sqlite3-Paket eingebunden. In Zeile 5 wird der Dateiname (inkl. Pfad) der Datenbank gebildet.
Zeilen 10 bis 23: Die Schrägstriche \ am Zeilenende besagen, dass die Programmzeile in der nächsten Zeile weiter geht. Die Programmzeile wurde nur der Übersichtlichkeit wegen umgebrochen. In Zeile 13 wird als Feldname `AlterJahre` genommen. Es wäre naheliegend das Feld `Alter` zu benennen. Dies führt aber zu einer Fehlermeldung, weil `alter` ein sqlite-Befehl ist. Deshalb wurde das Feld `AlterJahre` genannt. In den Zeilen 64 bis 81 wird eine Prozedur dargestellt, die zeigt, wie man Tcl/Tk-Befehle auf jede Ergebniszeile anwendet. In Zeile 70 wird eine Datenbankabfrage in folgender Form gemacht:
`db1 eval {select .....} Ergebnis {Tcl/Tk-Befehle}`

Wenn man hinter die Datenbankabfrage eine Variable schreibt und dahinter in geschweiften Klammern Tcl/Tk-Befehle, werden diese Befehle für jede Ergebniszeile ausgeführt: das Ergebnis aus der Datenbankabfrage wird sequentiell Zeile für Zeile in der Variablen `Ergebnis` gespeichert (es handelt sich um ein Array) und dann in den folgenden Anweisungen verwendet. Dies sieht man in Zeile 71. Der Befehl `parray` gibt den Inhalt der Variablen `Ergebnis` aus. Dies ist zunächst der erste Datensatz. In Zeile 72 wird der Inhalt der Spalte `AlterJahre` ausgewertet und der Personenzähler entsprechend erhöht. Anschließend wird der zweite Datensatz ausgewertet, dann der dritte usw. In den Zeilen 78 und 79 wird das Ergebnis der Zählung ausgegeben.
Sqlite3 entfaltet nur dann eine große Geschwindigkeit, wenn Transaktionen benutzt werden. Anstatt jeden sqlite-Befehl einzeln auszuführen, sollten die Befehle in einer Transaktion gesammelt und auf einmal ausgeführt werden. In Zeile 103 wird deshalb vor dem Einfügen der Adresse zunächst der Beginn einer Transaktion festgelegt. In Zeile 106 wird die Transaktion abgeschlossen, und jetzt erst führt sqlite die Befehle aus. Den Geschwindigkeitsunterschied beim Einfügen mit und ohne Transaktion kann man in der Abbildung sehen: das Einfügen von 300 Datensätzen dauert (auf meinem PC) ohne Transaktion 15,501 Sekunden, mit Transaktion nur 0,077 Sekunden.
In Zeile 136 wird die Datenbank-Datei gelöscht.
Wenn die Datenbank nicht in einer Datei gespeichert wird, sondern nur im Arbeitsspeicher gehalten wird, entfällt die Zeile 5. Außerdem muss dann die Zeile 34 geändert werden in `sqlite3 db1 ":memory:"`.

Listing 31.2: Dynamische Tabellennamen (Beispiel453.tcl)

```
1  #!/usr/bin/env tclsh
2
3  package require sqlite3
4
5  set Datenbank [file join [file dirname [info script]] ↩
       MeineDB] ; # entfaellt, wenn die Datenbank im ↩
       Arbeitsspeicher gehalten wird
6
7  proc DatenbankErstellen {} {
8    global db
9
10   for {set i 1} {$i <= 3} {incr i} {
11           db eval "create table tmpTabelle$i ( \
12                  ID integer primary key autoincrement, \
13                  Name text
14           )"
15   }
16 }
17
18 proc DatenbankOeffnen {} {
19   global Datenbank
20   global db
21   if {[file exist $Datenbank] == 0} {
22     # Datenbank existiert nicht
23     sqlite3 db $Datenbank ; # Alternativ: sqlite3 db "↩
          :memory:"
24     DatenbankErstellen
```

```
25    } else {
26      # Datenbank existiert
27      sqlite3 db $Datenbank
28    }
29  }
30
31  proc DatenbankSchliessen {} {
32    global db
33    catch db close
34  }
35
36  # Hauptprogramm
37
38  DatenbankOeffnen
39
40  db eval {insert into tmpTabelle1 values(NULL,"Anton")}
41  db eval {insert into tmpTabelle1 values(NULL,"Berta")}
42  db eval {insert into tmpTabelle2 values(NULL,"Caesar")}
43  db eval {insert into tmpTabelle3 values(NULL,"Dora")}
44
45  for {set i 1} {$i <= 3} {incr i} {
46          puts "Tabelle $i:"
47          puts [db eval "select * from tmpTabelle$i"]
48          puts "----"
49  }
50
51  DatenbankSchliessen
52
53  file delete -force $Datenbank
```



In den Zeilen 10 bis 15 werden drei Tabellen `tmpTabelle1` bis `tmpTabelle3` erstellt. Die Tabellennamen werden mit Hilfe der Variable `i` dynamisch erzeugt. Damit die Variable `i` durch die Zahlen 1 bis 3 ersetzt wird, muss der SQL-Befehl in Anführungszeichen gesetzt

werden (und nicht in geschweifte Klammern). In den Zeilen 45 bis 49 wird der Inhalt der drei Tabellen angezeigt. Auch hierbei müssen die SQL-Befehle in Anführungszeichen gesetzt werden, damit die Variable `i` durch die Zahlen 1 bis 3 ersetzt wird.

32 Objektorientierte Programmierung

Befehle:

- `oo::class create Klasse`
- `oo::define Klasse`
- `variable Variable`
- `constructor`
- `method Methode`
- `filter Methode`
- `export Methode`
- `unexport Methode`
- `Klasse create Objekt`
- `set Objekt [Klasse new]`
- `my`
- `next`
- `destructor`
- `Objekt destroy`
- `Klasse destroy`
- `superclass Klasse`
- `mixin Klasse`

Bisher stand die prozedurale Programmierung im Vordergrund. Nun soll die objektorientierte Programmierung in Tcl/Tk betrachtet werden. Der Unterschied zwischen beiden Programmierparadigmen wird an folgendem Beispiel recht deutlich: Bei der prozeduralen Programmierung erstellt man zum Beispiel eine Prozedur, die die Dateigröße anzeigt. Man ruft die Prozedur auf und übergibt ihr einen Dateinamen, nach dem Motto „Hallo Prozedur, zeige mir die Dateigröße für diese Datei“. Bei der objektorientierte Programmierung wechselt man die Sichtweise. Hier gibt es ein Datei-Objekt, das eine Funktion hat, die Dateigröße anzuzeigen. Der Aufruf erfolgt nach dem Motto „Hallo Datei, zeige mir deine Dateigröße“.

Neben diesem Wechsel der Perspektive bindet die objektorientierte Programmierung die Daten des Objekts (z. B. Vorname, Nachname, Geburtsdatum) und die Funktionen

(z. B. Zeige den Namen an, Berechne das Alter in Jahren) an das Objekt und kapselt sie gegenüber dem restlichen Programm. Der Zugriff auf die Daten des Objekts erfolgt dabei ausschließlich über objekteigene Funktionen. Die Funktionen nennt man Methoden.

Jedes Objekt gehört einer sogenannten Klasse an. Zum Beispiel könnte es eine Klasse „Fahrzeuge“ geben. Diese hat darunter die beiden Unterklassen „Motorisierte Fahrzeuge“ und „Nicht motorisierte Fahrzeuge“. Die Klasse der „Motorisierten Fahrzeuge“ hat die Unterklassen „PKW“, „LKW“ und „Motorrad“. Die Klasse der „PKW“ hat die Unterklassen „Audi“, „BMW“, „Citroen“ usw. Ein Objekt gehört bswp. der Klasse „Audi“ an.

Durch das Konzept der Vererbung kann man von einer übergeordneten Klasse Daten und Methoden auf die untergeordneten Klassen übertragen (vererben). Dadurch verfügt die untergeordnete Klasse über alle Daten und Methoden der übergeordneten Klasse und ergänzt diese um zusätzliche eigene Daten und Methoden. Z. B. hat die Klasse „Fahrzeuge“ die Eigenschaft „Anzahl Räder“. Somit haben alle davon abgeleiteten Klassen ebenfalls diese Eigenschaft. Zusätzlich hat die Klasse „Motorisierte Fahrzeuge“ die Eigenschaft „Art des Motors“.

Tabelle 32.1: Objektorientierte Befehle

Befehl	Beschreibung
`oo::class create Klasse`	Erzeugt eine Klasse, ohne sie zu definieren.
`oo::class create Klasse {}`	Erzeugt und definiert eine Klasse.
`oo::define Klasse {}`	Definiert eine bereits erzeugte Klasse.
`variable Variable`	Erzeugt eine Variable innerhalb der Klasse bzw. macht eine Variable der übergeordneten Klasse verfügbar. Sie wird dabei nicht mit einem Wert initialisiert.
`constructor {Parameter} {}`	Die `constructor`-Methode wird beim Erzeugen eines Objekts ausgeführt.
`method Methode {Parameter} {}`	Definiert eine Methode.
`export Methode`	Macht die Methode von außerhalb des Objekts zugänglich.
`unexport Methode`	Legt fest, dass die Methode nur innerhalb des Objekts aufgerufen werden kann.
`Klasse create Objekt`	Erzeugt ein Objekt einer bestimmten Klasse. Der Zugriff auf eine Methode des Objekts erfolgt mit `Objekt Methode`.

Tabelle 32.1: Objektorientierte Befehle

Befehl	Beschreibung
`Klasse create Objekt Parameter`	Erzeugt ein Objekt einer bestimmten Klasse.
`set Objekt [Klasse new Parameter]`	Erzeugt ein Objekt einer bestimmten Klasse (wie der vorherige Befehl). Es wird automatisch ein Objektname vergeben, den man in der Variablen Objekt speichert. Der Zugriff auf eine Methode des Objekts erfolgt mit `$Objekt Methode`.
`mixin Klasse`	Mixt eine weitere Klasse mit Methoden hinzu.
`my Methode`	Ruft eine Methode innerhalb des Objekts auf.
`my Variable`	Macht eine Variable der übergeordneten Klasse verfügbar, wenn sie nicht bereits mit dem Befehl `variable` verfügbar gemacht wurde. Sie wird dabei nicht mit einem Wert initialisiert.
`next`	Ruft die gleichnamige Methode der übergeordneten Klasse auf.
`next Parameter`	Ruft die gleichnamige Methode der übergeordneten Klasse auf und übergibt zusätzliche Parameter (wie beim Aufrufen einer Prozedur).
`filter Filtermethode`	Legt fest, welche Methode als Filter ausgeführt werden soll. Die Filtermethode wird nach der `constructor`-Methode aufgerufen, aber vor allen anderen Methoden.
`[self target]`	Gibt den Namen der Methode zurück, die nach der Filter-Methode ausgeführt wird.
`destructor {}`	Die `destructor`-Methode wird beim Löschen eines Objekts ausgeführt.

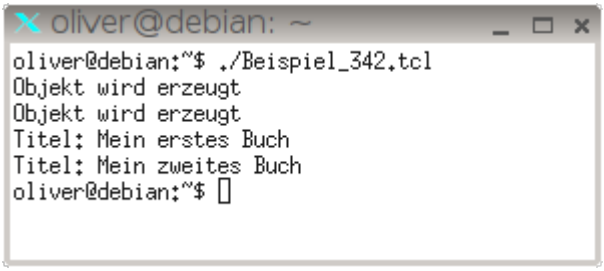
Tabelle 32.1: Objektorientierte Befehle

Befehl	Beschreibung
`Objekt destroy`	Löscht ein Objekt.
`Klasse destroy`	Löscht eine Klasse inklusive aller Unterklassen und zugehörigen Objekte.

Die folgenden Beispiele sollen eine Leihbücherei abbilden. Diese verleiht Bücher und DVDs, die beide zu der Klasse der `Verleihobjekte` gehören sollen.

Listing 32.1: Eine Klasse mit create erzeugen (Beispiel342.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt {
4          variable Titel
5
6          constructor {tmpTitel} {
7                  puts "Objekt wird erzeugt"
8                  set Titel $tmpTitel
9          }
10
11         method Anzeigen {} {
12                 puts "Titel: $Titel"
13         }
14
15         export Anzeigen
16 }
17
18 Verleihobjekt create Buch1 "Mein erstes Buch"
19 Buch1 Anzeigen
20
21 set Buch2 [Verleihobjekt new "Mein zweites Buch"]
22 $Buch2 Anzeigen
```



In Zeile 3 wird die Klasse `Verleihobjekt` erzeugt. In Zeile 4 wird festgelegt, dass es in dieser Klasse eine Variable `Titel` gibt. Diese Variable steht später allen abgeleiteten (Unter-)Klassen ebenfalls zur Verfügung.

Die Zeilen 6 bis 9 beinhalten die `constructor`-Methode. Jede Klasse hat immer eine `constructor`-Methode, auch wenn man sie nicht explizit benennt. Die Methode wird automatisch aufgerufen, sobald ein Objekt der Klasse zugeordnet wird, z. B. wenn ein Buch als Verleihobjekt erzeugt wird. Die `constructor`-Methode erwartet als Übergabewert einen Buch- oder DVD-Titel (Zeile 6), gibt danach einen Text aus (Zeile 7) und speichert den Titel in der Variablen `Titel` (Zeile 8). Man kann sich merken, dass der Aufbau der `constructor`-Methoden identisch ist mit dem Aufbau einer Prozeduren.

Die Zeilen 11 bis 13 umfassen die Methode `Anzeigen`. Es wird der Titel des Objekts ausgegeben (Zeile 12).

In Zeile 15 wird festgelegt, dass die Methode `Anzeigen` auch außerhalb des Objekts aufgerufen werden darf. Mit dem Befehl `unexport` würde man bestimmen, dass die Methode nur innerhalb des Objekts verfügbar ist. Achtung: es gibt bei Tcl/Tk folgende Automatik: Beginnt der Name der Methode mit einem Kleinbuchstaben, werden sie automatisch exportiert. Methoden, die mit einem Großbuchstaben beginnen, werden nicht automatisch exportiert. Um keine unerwarteten Überraschungen zu erleben und es dem Leser zu vereinfachen, sollte man alle Methoden immer mit `export` bzw. `unexport` kennzeichnen.

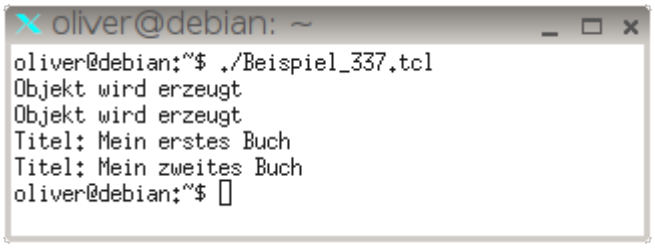
Es gibt zwei Möglichkeiten, Objekte zu erzeugen. Die erste Form ist in Zeile 18 zu sehen. Es wird ein `Buch1` als Objekt der Klasse `Verleihobjekt` erzeugt. Der Titel wird als Paramter an die `constructor`-Methode übergeben. In Zeile 19 wird die Methode `Anzeigen` des Objekts `Buch1` aufgerufen.

In Zeile 21 ist die zweite Form zu sehen. Es wird ein `Buch2` als Objekt der Klasse `Verleihobjekt` erzeugt. Dabei steht hinter dem Namen der Klasse der Befehl `new`, gefolgt vom Titel des Buchs. Der Titel wird als Paramter an die `constructor`-Methode übergeben. Als Rückgabe bekommt man einen Verweis auf das Objekt, der in die Variable `Buch2` gespeichert wird. In Zeile 22 wird die Methode `Anzeigen` des Objekts `Buch2` aufgerufen. Wie man sieht greift man auf das Objekt `Buch2` genauso zu wie man auf eine Variable zugreift, nämlich mit einem vorangestellten `$`-Zeichen.

Listing 32.2: Eine Klasse mit create und define erzeugen (Beispiel337.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt
4
5  oo::define Verleihobjekt {
6          variable Titel
7
8          constructor {tmpTitel} {
9                  puts "Objekt wird erzeugt"
10                 set Titel $tmpTitel
11         }
12
13         method Anzeigen {} {
14                 puts "Titel: $Titel"
15         }
16
17         export Anzeigen
18 }
19
```

```
20 set Buch1 [Verleihobjekt new "Mein erstes Buch"]
21 set Buch2 [Verleihobjekt new "Mein zweites Buch"]
22 $Buch1 Anzeigen
23 $Buch2 Anzeigen
```



In Zeile 3 wird eine leere Klasse `Verleihobjekt` erzeugt. Die Klasse hat noch keine Variablen und Methoden. Mit Hilfe des `define`-Befehls kann man alle Elemente einer Klasse (Variablen, Methoden, Filter usw.) verändern, löschen und hinzufügen. In diesem Beispiel umfasst der `define`-Befehl alle Befehle, die im vorherigen Beispiel im `create`-Befehl standen. Es ist empfehlenswert, Klassen nur mit dem `create`-Befehl zu erzeugen und den `define`-Befehl nur für spätere Änderungen während der Programmausführung zu verwenden.

Listing 32.3: Objekt erzeugen mit create bzw. new (Beispiel343.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt {
4    variable Titel
5
6    constructor {tmpTitel} {
7      puts "Objekt wird erzeugt"
8      set Titel $tmpTitel
9    }
10
11   method Anzeigen {} {
12     puts "Titel: $Titel"
13   }
14
15   export Anzeigen
16 }
17
18 Verleihobjekt create Buch1 "Mein erstes Buch"
19 Buch1 Anzeigen
20
21 set Buch2 [Verleihobjekt new "Mein zweites Buch"]
22 $Buch2 Anzeigen
```

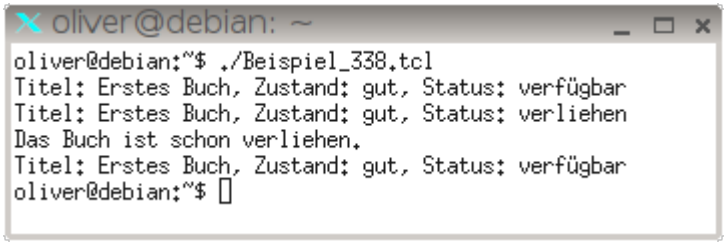
```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_343.tcl
Objekt wird erzeugt
Titel: Mein erstes Buch
Objekt wird erzeugt
Titel: Mein zweites Buch
oliver@debian:~$
```

In Zeile 18 wird das Objekt `Buch1` mit dem Befehl `create` erzeugt. Der Befehl hat die Form `Klasse create Objekt Paramter`. Wenn man auf diese Weise ein Objekt erzeugt, bekommt das Objekt den angegebenen Namen und kann an Hand dieses Names im weiteren Programm angesprochen werden (wie in Zeile 19 zu sehen).
In Zeile 20 wird das Objekt `Buch2` mit dem Befehl `new` erzeugt. Die Form lautet `[Klasse new Parameter]` Der Rückgabewert ist ein interner Objektname, den man in einer Variablen speichert. Der Zugriff auf das Objekt erfolgt, in dem man den Inhalt der Variablen `$Buch2` verwendet (Zeile 22).

Listing 32.4: Methoden aufrufen (Beispiel338.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt {
4          variable Titel
5          variable Zustand
6          variable Verleihstatus
7
8          constructor {tmpTitel tmpZustand} {
9                  set Titel $tmpTitel
10                 set Zustand $tmpZustand
11                 set Verleihstatus "verfuegbar"
12         }
13
14         method Anzeigen {} {
15                 puts "Titel: $Titel, Zustand: $Zustand, ↩
                      Status: $Verleihstatus"
16         }
17
18         method Ausleihen {} {
19                 if {$Verleihstatus == "verfuegbar"} {
20                         set Verleihstatus "verliehen"
21                 } else {
22                         puts "Das Buch ist schon ↩
                              verliehen."
23                 }
24         }
25
26         method Zurueckgeben {} {
27                 if {$Verleihstatus == "verliehen"} {
28                         set Verleihstatus "verfuegbar"
29                 }
30         }
31
```

```
32          export Anzeigen
33          export Ausleihen
34          export Zurueckgeben
35 }
36
37 set Buch1 [Verleihobjekt new "Erstes Buch" "gut"]
38
39 $Buch1 Anzeigen
40 $Buch1 Ausleihen
41 $Buch1 Anzeigen
42 $Buch1 Ausleihen
43 $Buch1 Zurueckgeben
44 $Buch1 Anzeigen
```



Dieses Beispiel erweitert das vorangegangene Beispiel. In den Zeilen 4 bis 6 werden die drei Variablen `Titel`, `Zustand` und `Verleihobjekt` definiert. Die `constructor`-Methode in Zeile 8 erwartet zwei Parameter: einen Titel und einen Zustand. In den Zeilen 18 bis 24 wird die Methode `Ausleihen` definiert. In den Zeilen 26 bis 30 wird die Methode `Zurueckgeben` festgelegt. In den Zeilen 33 und 34 werden die beiden neuen Methoden `Ausleihen` und `Zurueckgeben` exportiert, so dass sie auch verfügbar sind. In Zeile 37 werden beim Erzeugen des Objekts `Buch1` die beiden Parameter `Titel` und `Zustand` übergeben. In den Zeilen 39 bis 44 werden die verschiedenen Methoden von `Buch1` aufgerufen.

Listing 32.5: Eine Methode ruft eine andere Methode auf (Beispiel339.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt {
4    variable Titel
5    variable Zustand
6    variable Verleihstatus
7
8    constructor {tmpTitel tmpZustand} {
9      set Titel $tmpTitel
10     set Zustand $tmpZustand
11     set Verleihstatus "verfuegbar"
12   }
13
14   method FehlerAnzeigen {Text} {
15     puts $Text
16   }
17
```

```
18    method Anzeigen {} {
19      puts "Titel: $Titel, Zustand: $Zustand, Status: ↩
          $Verleihstatus"
20    }
21
22    method Ausleihen {} {
23      if {$Verleihstatus == "verfuegbar"} {
24        set Verleihstatus "verliehen"
25      } else {
26        my FehlerAnzeigen "Das Buch ist schon verliehen."
27      }
28    }
29
30    method Zurueckgeben {} {
31      if {$Verleihstatus == "verliehen"} {
32        set Verleihstatus "verfuegbar"
33      }
34    }
35
36    unexport FehlerAnzeigen
37    export Anzeigen
38    export Ausleihen
39    export Zurueckgeben
40  }
41
42  set Buch1 [Verleihobjekt new "Erstes Buch" "gut"]
43
44  $Buch1 Anzeigen
45  $Buch1 Ausleihen
46  $Buch1 Anzeigen
47  $Buch1 Ausleihen
48  $Buch1 Zurueckgeben
49  $Buch1 Anzeigen
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_338.tcl
Titel: Erstes Buch, Zustand: gut, Status: verfügbar
Titel: Erstes Buch, Zustand: gut, Status: verliehen
Das Buch ist schon verliehen.
Titel: Erstes Buch, Zustand: gut, Status: verfügbar
oliver@debian:~$
```

In Zeilen 14 bis 16 wird die Methode `FehlerAnzeigen` definiert. Diese Methode soll die Fehlermeldungen zum Objekt ausgeben. In Zeile 26 wird die Fehlermeldung nicht (wie im Beispiel zuvor) direkt ausgegeben, sondern es wird ein Text an die Methode `FehlerAnzeigen` übergeben. Da diese Methode innerhalb des Objekts existiert, muss man das Schlüsselwort `my` voranstellen, das auf das eigene Objekt verweist. In Zeile 36 wird festgelegt, das die Methode `FehlerAnzeigen` nicht von außerhalb des Objekts aufgerufen werden kann.

Ein Objekt kapselt die Daten (Eigenschaften) und Methoden und trennt sie vom

restlichen Programm. Wenn man vom Hauptprogramm aus die Daten des Objekts ändern will, muss das Objekt entsprechende Methoden anbieten.

Listing 32.6: Daten/Eigenschaften eines Objekts ändern (Beispiel340.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt {
4    variable Titel
5    variable Zustand
6    variable Verleihstatus
7
8    constructor {tmpTitel tmpZustand} {
9      set Titel $tmpTitel
10     set Zustand $tmpZustand
11     set Verleihstatus "verfuegbar"
12   }
13
14   method Anzeigen {} {
15     puts "Titel: $Titel, Zustand: $Zustand, Status: ↩
         $Verleihstatus"
16   }
17
18   method TitelAendern {NeuerTitel} {
19     set Titel $NeuerTitel
20   }
21
22   export Anzeigen
23   export TitelAendern
24 }
25
26 set Buch1 [Verleihobjekt new "Erstes Buch" "gut"]
27
28 $Buch1 Anzeigen
29 $Buch1 TitelAendern "Mein Buch"
30 $Buch1 Anzeigen
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_340.tcl
Titel: Erstes Buch, Zustand: gut, Status: verfügbar
Titel: Mein Buch, Zustand: gut, Status: verfügbar
oliver@debian:~$
```

In den Zeilen 18 bis 20 wird die Methode `TitelAendern` zum Ändern des Titels definiert. Diese Methode erwartet als Übergabewert einen neuen Titel. Nur über diese Methode kann man den Buchtitel ändern. In Zeile 23 wird die Methode exportiert. In Zeile 29 wird die Methode aufgerufen und der Buchtitel geändert.

Listing 32.7: Vererbung (Beispiel341.tcl)

```
1  #!/usr/bin/env tclsh
2
```

```
3  oo::class create Verleihobjekt {
4    puts "Variablen im Verleihobjekt"
5    variable Titel
6    variable Zustand
7    variable Verleihstatus
8
9    constructor {tmpTitel tmpZustand} {
10     puts "Constructor im Verleihobjekt wird ausgefuehrt."
11     set Titel $tmpTitel
12     set Zustand $tmpZustand
13     set Verleihstatus "verfuegbar"
14   }
15
16   method Anzeigen {} {
17     puts "Anzeigen:"
18     puts "Titel: $Titel, Zustand: $Zustand, Status: ↩
          $Verleihstatus"
19   }
20
21   export Anzeigen
22 }
23
24 oo::class create Buchobjekt {
25   superclass Verleihobjekt
26
27   puts "Variablen aus der uebergeordneten Klasse ↩
        verfuegbar machen:"
28   variable Titel
29   variable Zustand
30
31   puts "Variablen im Buchobjekt"
32   variable Autor
33   variable ISBN
34
35   constructor {tmpTitel tmpZustand tmpAutor tmpISBN} {
36     next $tmpTitel $tmpZustand
37     puts "Constructor im Buchobjekt wird ausgefuehrt."
38     set Autor $tmpAutor
39     set ISBN $tmpISBN
40   }
41
42   method BuchAnzeigen1 {} {
43     puts "BuchAnzeigen1:"
44     my variable Verleihstatus
45     puts "Titel: $Titel, Zustand: $Zustand, Status: ↩
          $Verleihstatus, Autor: $Autor, ISBN: $ISBN"
46   }
47
48   method BuchAnzeigen2 {} {
49     puts "BuchAnzeigen2:"
50     my Anzeigen
51     puts "Autor: $Autor, ISBN: $ISBN"
52   }
```

```
53
54    export BuchAnzeigen1
55    export BuchAnzeigen2
56 }
57
58 set Buch1 [Buchobjekt new "Erstes Buch" "gut" "Scholl" "
      123-456-789"]
59
60 $Buch1 Anzeigen
61 $Buch1 BuchAnzeigen1
62 $Buch1 BuchAnzeigen2
```

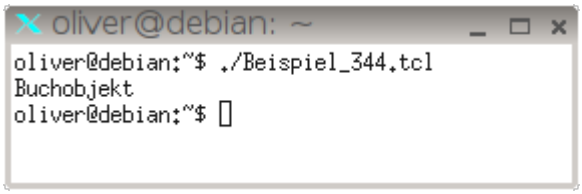
```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_341.tcl
Variablen im Verleihobjekt
Variablen aus der übergeordneten Klasse verfügbar machen:
Variablen im Buchobjekt
Constructor im Verleihobjekt wird ausgeführt.
Constructor im Buchobjekt wird ausgeführt.
Anzeigen:
Titel: Erstes Buch, Zustand: gut, Status: verfügbar
BuchAnzeigen1:
Titel: Erstes Buch, Zustand: gut, Status: verfügbar, Autor: Scholl, ISBN: 123-456-789
BuchAnzeigen2:
Anzeigen:
Titel: Erstes Buch, Zustand: gut, Status: verfügbar
Autor: Scholl, ISBN: 123-456-789
oliver@debian:~$
```

In den Zeilen 3 bis 22 wird die Hauptklasse `Verleihobjekt` definiert. In den Zeilen 24 bis 56 wird die untergeordnete Klasse `Buchobjekt` definiert. Sie soll alle Variablen und Methoden aus der Hauptklasse erben und zusätzlich über weitere, eigene Variablen (`Autor`, `ISBN`) und eigene Methoden (`BuchAnzeigen1`, `BuchAnzeigen2`) verfügen. In Zeile 25 erbt das Buchobjekt durch den Befehl `superclass` alle Variablen und Methoden aus der Hauptklasse `Verleihobjekt`. In den Zeilen 28 und 29 werden die beiden Variablen `Titel` und `Zustand` aus der Hauptklasse verfügbar gemacht. Die Variable `Verleihstatus` aus der Hauptklasse wird absichtlich noch nicht verfügbar gemacht, um später in Zeile 44 einen alternativen Zugriff zu zeigen. In den Zeilen 32 und 33 werden weitere Variablen definiert, die nur im Buchobjekt vorkommen.
In Zeile 36 wird mit dem Befehl `next` die gleichnamige Methode in der Hauptklasse aufgerufen, also die `constructor`-Methode. Diese verlangt die beiden Parameter `Titel` und `Zustand`. Bei der Abarbeitung der `constructor`-Methode des Buchobjekts springt das Programm in die Hauptklasse (Zeile 36), führt deren `constructor`-Methode aus (Zeilen 9 bis 14) und setzt dann im Buchobjekt die `constructor`-Methode fort (Zeile 37). In den Zeilen 38 und 39 werden danach weitere Daten festgelegt, die es nur im Buchobjekt gibt.
In Zeile 44 wird die Variable `Verleihstatus` aus dem Hauptobjekt verfügbar gemacht. In Zeile 50 wird die aus der Hauptklasse geerbte Methode `Anzeigen` aufgerufen. Üblicherweise ruft man eine Methode mit der Syntax `Objekt Methode` (z. B. `Buch1 Anzeigen`)

auf. Wenn man eine Methode im eigenen Objekt aufrufen möchte, schreibt man statt des Objekts das Schlüsselwort `my` (z. B. `my Anzeigen`).

Listing 32.8: Gleicher Methodennamen in Haupt- und Unterklasse (Variante 1) (Beispiel344.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt {
4    variable Titel
5
6    constructor {tmpTitel} {
7      set Titel $tmpTitel
8    }
9
10   method Anzeigen {} {
11     puts "Verleihobjekt"
12   }
13
14   export Anzeigen
15 }
16
17 oo::class create Buchobjekt {
18   superclass Verleihobjekt
19
20   method Anzeigen {} {
21     puts "Buchobjekt"
22   }
23
24   export Anzeigen
25 }
26
27 Buchobjekt create Buch1 "Mein erstes Buch"
28 Buch1 Anzeigen
```



In den Zeilen 10 bis 12 wird in der Hauptklasse `Verleihobjekt` die Methode `Anzeigen` definiert. In den Zeilen 20 bis 22 wird in der Unterklasse `Buchobjekt` ebenfalls eine Methode `Anzeigen` festgelegt. In Zeile 28 wird danach die Methode `Anzeigen` aufgerufen. Wie man am Ergebnis sieht, wird dadurch nur die Methode der Unterklasse `Buchobjekt` ausgeführt.

Listing 32.9: Gleicher Methodennamen in Haupt- und Unterklasse (Variante 2) (Beispiel345.tcl)

```
1  #!/usr/bin/env tclsh
2
```

```
3  oo::class create Verleihobjekt {
4    variable Titel
5
6    constructor {tmpTitel} {
7      set Titel $tmpTitel
8    }
9
10   method Anzeigen {} {
11     puts "Verleihobjekt"
12   }
13
14   export Anzeigen
15 }
16
17 oo::class create Buchobjekt {
18   superclass Verleihobjekt
19
20   method Anzeigen {} {
21     next
22     puts "Buchobjekt"
23     next
24   }
25
26   export Anzeigen
27 }
28
29 Buchobjekt create Buch1 "Mein erstes Buch"
30 Buch1 Anzeigen
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_345.tcl
Verleihobjekt
Buchobjekt
oliver@debian:~$
```

In den Zeilen 21 und 23 wurde der Befehl `next` eingefügt. In Zeile 30 wird die Methode `Anzeigen` im `Buchobjekt` aufgerufen. Diese Methode springt durch den `next`-Befehl (Zeile 21) in die gleichnamige Methode der Hauptklasse `Verleihobjekt`. Danach wird die Methode `Anzeigen` in der Unterklasse `Buchobjekt` weiter ausgeführt (Zeile 22). In Zeile 23 erfolgt erneut der Aufruf der Methode `Anzeigen` der Hauptklasse `Verleihobjekt`.

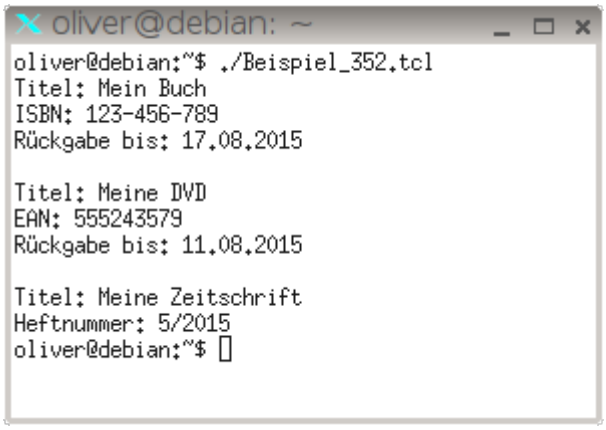
Angenommen die Bücherei hat neben Büchern und DVDs auch noch Zeitschriften, die man allerdings nicht ausleihen darf. Dann wäre eine Methode `RueckgabeterminSetzen` nur bei Büchern und DVDs sinnvoll, nicht aber bei Zeitschriften. Um diese Methode nicht in den beiden Klassen `Buch` und `DVD` doppelt zu programmieren, kann man die Methode als eigene Klasse anlegen und mit dem `mixin`-Befehl dazuladen. Wenn man sich die Vererbung wie einen senkrechten Stammbaum vorstellt, dann ergänzt `mixin` weitere Methoden waagrecht, so dass man eine Art Matrix erhält.

Listing 32.10: Mix-in von Methoden (Beispiel352.tcl)

```
#!/usr/bin/env tclsh

oo::class create KlasseBuecherei {
  variable Titel

  constructor {tmpTitel} {
    set Titel $tmpTitel
  }

  method Anzeigen {} {
    puts "Titel: $Titel"
  }

  export Anzeigen
}

oo::class create KlasseAusleihe {
  variable Rueckgabedatum

  method SetzeRueckgabedatum {tmpDatum} {
    set Rueckgabedatum $tmpDatum
  }

  method ZeigeRueckgabedatum {} {
    puts "Rueckgabe bis: $Rueckgabedatum"
  }

  export SetzeRueckgabedatum
  export ZeigeRueckgabedatum
}

oo::class create KlasseBuch {
  superclass KlasseBuecherei
  mixin KlasseAusleihe

  variable Titel
  variable ISBN

  method SetzeISBN {tmpISBN} {
    set ISBN $tmpISBN
  }

  method Anzeigen {} {
    next
    puts "ISBN: $ISBN"
        my ZeigeRueckgabedatum
  }

  export SetzeISBN
  export Anzeigen
}
```

```
52
53 oo::class create KlasseDVD {
54   superclass KlasseBuecherei
55   mixin KlasseAusleihe
56
57   variable Titel
58   variable EAN
59
60   method SetzeEAN {tmpEAN} {
61     set EAN $tmpEAN
62   }
63
64   method Anzeigen {} {
65     next
66     puts "EAN: $EAN"
67         my ZeigeRueckgabedatum
68   }
69
70   export SetzeEAN
71   export Anzeigen
72 }
73
74 oo::class create KlasseZeitschrift {
75   superclass KlasseBuecherei
76
77   variable Titel
78   variable Heftnummer
79
80   method SetzeHeftnummer {tmpHeftnummer} {
81     set Heftnummer $tmpHeftnummer
82   }
83
84   method Anzeigen {} {
85     next
86     puts "Heftnummer: $Heftnummer"
87   }
88
89   export SetzeHeftnummer
90   export Anzeigen
91 }
92
93 KlasseBuch create Buch1 "Mein Buch"
94 KlasseDVD create DVD1 "Meine DVD"
95 KlasseZeitschrift create Zeitschrift1 "Meine Zeitschrift"
96
97 Buch1 SetzeISBN "123-456-789"
98 DVD1 SetzeEAN "555243579"
99 Zeitschrift1 SetzeHeftnummer "5/2015"
100
101 Buch1 SetzeRueckgabedatum "17.08.2015"
102 DVD1 SetzeRueckgabedatum "11.08.2015"
103
104 Buch1 Anzeigen
```

```
105 puts ""
106 DVD1 Anzeigen
107 puts ""
108 Zeitschrift1 Anzeigen
```

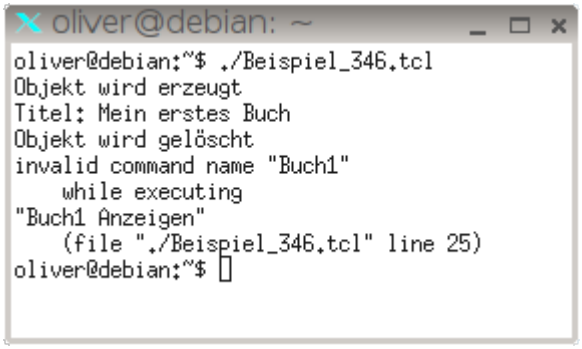


In den Zeilen 17 bis 30 wird eine Klasse `KlasseAusleihe` definiert, die zwei Methoden hat. Diese Methoden sollen nur den Klassen `KlasseBuch` und `KlasseDVD` zugeordnet werden, nicht aber der Klasse `KlasseZeitschrift`. In Zeile 34 werden die Methoden aus der Klasse `KlasseAusleihe` in die Klasse `KlasseBuch` importiert (hineingemixt). Dasselbe erfolgt für die Klasse `KlasseDVD` in Zeile 55. In den Zeilen 46 und 67 wird mit dem Befehl `my ZeigeRueckgabedatum` die hineingemixte Methode `ZeigeRueckgabedatum` benutzt. In den Zeilen 101 und 102 wird die andere hineingemixte Methode `SetzeRueckgabedatum` aufgerufen.

Listing 32.11: Objekt löschen (Beispiel346.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt {
4    variable Titel
5
6    constructor {tmpTitel} {
7      puts "Objekt wird erzeugt"
8      set Titel $tmpTitel
9    }
10
11   destructor {
12     puts "Objekt wird geloescht"
13   }
14
15   method Anzeigen {} {
16     puts "Titel: $Titel"
17   }
18
19   export Anzeigen
20 }
21
22 Verleihobjekt create Buch1 "Mein erstes Buch"
```

```
23 Buch1 Anzeigen
24 Buch1 destroy
25 Buch1 Anzeigen
```



In den Zeilen 11 bis 13 wird die `destructor`-Methode definiert. Diese Methode ist optional, muss also nicht definiert werden. Wenn die `destructor`-Methode vorhanden ist, wird sie immer ausgeführt, wenn ein Objekt gelöscht wird. In Zeile 24 wird das Objekt `Buch1` gelöscht. Da anschließend das Objekt nicht mehr existiert, kommt es in der nächsten Zeile (Zeile 25) zu einer Fehlermeldung.

Listing 32.12: Klasse löschen (Beispiel347.tcl)

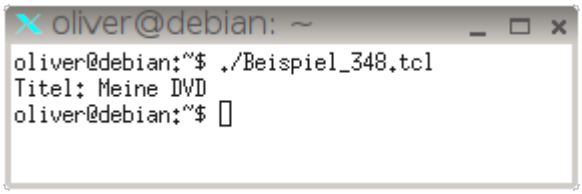
```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt {
4    variable Titel
5
6    constructor {tmpTitel} {
7      puts "Objekt wird erzeugt"
8      set Titel $tmpTitel
9    }
10
11   destructor {
12     puts "Objekt wird geloescht"
13   }
14
15   method Anzeigen {} {
16     puts "Titel: $Titel"
17   }
18
19   export Anzeigen
20 }
21
22 Verleihobjekt create Buch1 "Mein erstes Buch"
23 Buch1 Anzeigen
24 Verleihobjekt destroy
25 Buch1 Anzeigen
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_347.tcl
Objekt wird erzeugt
Titel: Mein erstes Buch
Objekt wird gelöscht
invalid command name "Buch1"
    while executing
"Buch1 Anzeigen"
    (file "./Beispiel_347.tcl" line 25)
oliver@debian:~$
```

In Zeile 24 wird die Klasse `Verleihobjekt` gelöscht. Dadurch werden auch alle Unterklassen und alle zugehörigen Objekte gelöscht.

Listing 32.13: Aus einem Objekt heraus die Methode eines anderen Objekts aufrufen (Beispiel348.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Verleihobjekt {
4    variable Titel
5
6    constructor {tmpTitel} {
7      set Titel $tmpTitel
8    }
9
10 }
11
12 oo::class create Buchobjekt {
13   superclass Verleihobjekt
14   variable Titel
15
16   method DVDAnzeigen {tmpObjekt} {
17     $tmpObjekt Anzeigen
18   }
19
20   export DVDAnzeigen
21 }
22
23 oo::class create DVDobjekt {
24   superclass Verleihobjekt
25   variable Titel
26
27   method Anzeigen {} {
28     puts "Titel: $Titel"
29   }
30
31   export Anzeigen
32 }
33
34 Buchobjekt create Buch1 "Mein Buch"
35 DVDobjekt create DVD1 "Meine DVD"
36 Buch1 DVDAnzeigen DVD1
```



In den Zeilen 12 bis 21 wird das `Buchobjekt` definiert. Es hat eine Methode `DVDAnzeigen` (Zeile 16), die als Parameter ein (DVD-)Objekt erwartet. In Zeile 17 wird die Methode `Anzeigen` des DVD-Objekts aufgerufen. In den Zeilen 23 bis 32 wird das `DVDobjekt` definiert. In Zeile 34 wird das Objekt `Buch1` erzeugt, in Zeile 35 das Objekt `DVD1`. In Zeile 36 wird die Methode `DVDAnzeigen` des Objekts `Buch1` aufgerufen und der Name des DVD-Objekts übergeben. Im Ergebnis sieht man, dass der Titel der DVD ausgegeben wird.

Angenommen, man möchte die Fläche verschiedener Figuren (Rechteck, Kreis usw.) berechnen. In der prozeduralen Programmierung muss eine solche Prozedur `FlaecheBerechnen` eine Fallunterscheidung nach der Art der Figur vornehmen. Bei der objektorientierten Programmierung definiert man zu jeder Figur eine Methode `FlaecheBerechnen`. Wenn man dann in einer Schleife die Fläche verschiedener Figuren berechnet, weiß jedes Objekt selbst, wie seine Fläche berechnet wird.

Listing 32.14: Gleiche Methode bei verschiedenen Objekten ausführen (Beispiel349.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Figur {
4    variable Name
5    variable Flaeche
6
7    constructor {tmpName} {
8      set Name $tmpName
9      set Flaeche "unbekannt"
10   }
11
12   method Anzeigen {} {
13     puts "$Name: Flaeche=$Flaeche"
14   }
15
16   export Anzeigen
17 }
18
19 oo::class create Rechteck {
20   superclass Figur
21
22   variable Flaeche
23   variable SeiteA
24   variable SeiteB
25
26   constructor {tmpName tmpSeiteA tmpSeiteB} {
27     next $tmpName
```

```
28      set SeiteA $tmpSeiteA
29      set SeiteB $tmpSeiteB
30    }
31
32    method Flaeche {} {
33      set Flaeche [expr 1.0 * $SeiteA * $SeiteB]
34    }
35
36    export Flaeche
37  }
38
39  oo::class create Kreis {
40    superclass Figur
41
42    variable Flaeche
43    variable Radius
44
45    constructor {tmpName tmpRadius} {
46      next $tmpName
47      set Radius $tmpRadius
48    }
49
50    method Flaeche {} {
51      set Flaeche [expr 3.141 * $Radius * $Radius]
52    }
53
54    export Flaeche
55  }
56
57  Rechteck create Rechteck1 "Rechteck1" 3 5
58  Rechteck1 Anzeigen
59  Rechteck1 Flaeche
60  Rechteck1 Anzeigen
61
62  Kreis create Kreis1 "Kreis1" 8
63  Kreis1 Anzeigen
64  Kreis1 Flaeche
65  Kreis1 Anzeigen
66
67  puts ""
68  puts "jetzt als Schleife:"
69  set Liste {Rechteck1 Kreis1}
70  foreach Element $Liste {
71    $Element Flaeche
72    $Element Anzeigen
73  }
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_349.tcl
Rechteck1: Fläche=unbekannt
Rechteck1: Fläche=15.0
Kreis1: Fläche=unbekannt
Kreis1: Fläche=201.024

jetzt als Schleife:
Rechteck1: Fläche=15.0
Kreis1: Fläche=201.024
oliver@debian:~$
```

In den Zeilen 3 bis 17 wird die Hauptklasse `Figur` definiert. Sie beinhaltet auch die Methode `Anzeigen`, die allen abgeleiteten Klassen zur Verfügung stehen soll. In den Zeilen 19 bis 37 wird die Klasse `Rechteck` festgelegt. Sie hat eine eigene Methode `Flaeche` zur Berechnung der Fläche. In den Zeilen 39 bis 55 wird die Klasse `Kreis` definiert, die ebenfalls eine eigene Methode `Flaeche` hat. In Zeile 57 wird eine Figur `Rechteck1` erzeugt. Dessen Fläche ist noch nicht berechnet (Ausgabe gemäß Zeile 58). In Zeile 59 wird die Fläche des Rechtecks berechnet und angezeigt (Zeile 60). In den Zeilen 62 bis 65 geschieht das genauso für einen Kreis.
In Zeile 69 wird eine Liste mit mehreren Objekten erstellt. In den Zeilen 70 bis 73 wird diese Liste durchlaufen und zu jedem Objekt die Fläche berechnet. Wie man sieht wird bei allen Objekten die (namentlich) gleiche Methode `Flaeche` aufgerufen. Die Objekte berechnen aber individuell ihre Fläche.

Ein Filter ist eine Methode, die vor jeder anderen Methode ausgeführt. Nur bei der `constructor`-Methode und der `destructor`-Methode wird die Filter-Methode nicht ausgeführt. Mit einem Filter kann man z. B. die Zulässigkeit von Werten überprüfen, bevor andere Methoden ausgeführt werden. Der Filter hat die gleiche Syntax wie eine Methode. In einem Filter kann man mit dem `return`-Befehl die weitere Verarbeitung beenden, so dass die eigentlich aufgerufene Methode nicht ausgeführt wird. Man kann aber auch in der Filter-Methode die überprüften Werte korrigieren und dann mit dem `next`-Befehl die eigentliche Methode ausführen. Den Namen, der eigentlich aufgerufenen Methode erhält man mit dem Befehl `[self target]`.

Man darf auch mehrere Filter festlegen, die dann der Reihe nach ausgeführt werden. Außerdem kann man den/die Filter als eigene Klasse erstellen, die man mit dem Befehl `mixin Filterklasse` importiert und dann mit dem Befehl `filter Filtermethode` aufruft.

Wenn man an die eigentliche Methode Parameter übergeben will, müssen diese vom Filter weitergeleitet werden. Das erreicht man, in dem der Filter die Argumente mit `{args}`entgegennimmt und mit `return [next {*}$args]` weitergibt.

Listing 32.15: Filter verhindert das Ausführen einer Methode (Beispiel350.tcl)

```
1 #!/usr/bin/env tclsh
2
3 oo::class create Figur {
4   variable Name
5   variable Flaeche
6
```

```
7   constructor {tmpName} {
8     set Name $tmpName
9     set Flaeche "unbekannt"
10  }
11
12  method Anzeigen {} {
13    puts "$Name: Flaeche=$Flaeche"
14  }
15
16  export Anzeigen
17 }
18
19 oo::class create Rechteck {
20  superclass Figur
21
22  variable Flaeche
23  variable SeiteA
24  variable SeiteB
25
26  filter WertePruefen
27
28  constructor {tmpName tmpSeiteA tmpSeiteB} {
29    next $tmpName
30    set SeiteA $tmpSeiteA
31    set SeiteB $tmpSeiteB
32  }
33
34  method WertePruefen {} {
35    set Methode [self target]
36    puts "Die aufgerufene Methode ist $Methode"
37    if {$SeiteA < 0 || $SeiteB < 0} {
38          puts "Die Werte sind unzulaessig."
39          return
40    }
41    next
42  }
43
44  method Flaeche {} {
45    set Flaeche [expr 1.0 * $SeiteA * $SeiteB]
46  }
47
48  export Flaeche
49  unexport WertePruefen
50 }
51
52 Rechteck create Rechteck1 "Rechteck1" -3 5
53 Rechteck1 Anzeigen
54 Rechteck1 Flaeche
```

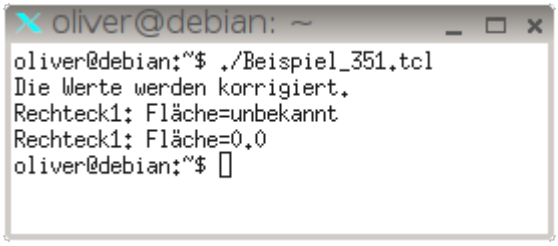
```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_350.tcl
Die aufgerufene Methode ist ::Figur Anzeigen
Die Werte sind unzulässig.
Die aufgerufene Methode ist ::Rechteck Flaeche
Die Werte sind unzulässig.
oliver@debian:~$
```

In Zeile 26 wird festgelegt, dass der Filter `WertePruefen` ausgeführt werden soll. Der Filter steht in den Zeilen 34 bis 41. Er überprüft, ob die Werte kleiner als Null sind. Falls dies der Fall ist, wird der Aufruf der weiteren Methode durch den Befehl `return` (Zeile 39) verhindert. In Zeile 35 wird der Name der Methode ermittelt, die nach dem Filter aufgerufen wird. In Zeile 41 wird mit dem Befehl `next` die Ziel-Methode `Anzeigen` bzw. `Flaeche` aufgerufen. In Zeile 49 wird festgelegt, dass die Filter-Methode `WertePruefen` nur innerhalb der Klasse aufgerufen werden kann. Wie man an der Ausgabe sieht, werden durch den Filter alle Methoden-Aufrufe blockiert. Wenn man aber nur die Methode `Flaeche` blockieren will, muss man in dieser Methode mit einer if-Bedingung die Berechnung verhindern.

Listing 32.16: Filter korrigiert die Werte und führt dann die Methode aus (Beispiel351.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Figur {
4    variable Name
5    variable Flaeche
6
7    constructor {tmpName} {
8      set Name $tmpName
9      set Flaeche "unbekannt"
10   }
11
12   method Anzeigen {} {
13     puts "$Name: Flaeche=$Flaeche"
14   }
15
16   export Anzeigen
17 }
18
19 oo::class create Rechteck {
20   superclass Figur
21
22   variable Flaeche
23   variable SeiteA
24   variable SeiteB
25
26   filter WertePruefen
27
28   constructor {tmpName tmpSeiteA tmpSeiteB} {
29     next $tmpName
```

```
30      set SeiteA $tmpSeiteA
31      set SeiteB $tmpSeiteB
32    }
33
34    method WertePruefen {} {
35      if {$SeiteA < 0 || $SeiteB < 0} {
36            puts "Die Werte werden korrigiert."
37            set SeiteA 0
38            set SeiteB 0
39          }
40          next
41    }
42
43    method Flaeche {} {
44      set Flaeche [expr 1.0 * $SeiteA * $SeiteB]
45    }
46
47    export Flaeche
48    unexport WertePruefen
49  }
50
51  Rechteck create Rechteck1 "Rechteck1" -3 5
52  Rechteck1 Anzeigen
53  Rechteck1 Flaeche
54  Rechteck1 Anzeigen
```



In den Zeilen 37 und 38 werden im Unterschied zum vorherigen Beispiel die Werte für `SeiteA` und `SeiteB` geändert, wenn einer der beiden Werte kleiner als Null ist.

Listing 32.17: Filter mit dem Befehl mixin importieren (Beispiel353.tcl)

```
1   #!/usr/bin/env tclsh
2
3   oo::class create Figur {
4     variable Name
5     variable Flaeche
6
7     constructor {tmpName} {
8       set Name $tmpName
9       set Flaeche "unbekannt"
10    }
11
12    method Anzeigen {} {
13      puts "$Name: Flaeche=$Flaeche"
```

```
14    }
15
16    export Anzeigen
17  }
18
19  oo::class create Filter {
20    variable SeiteA
21    variable SeiteB
22
23    method WertePruefen {} {
24      if {$SeiteA < 0 || $SeiteB < 0} {
25            puts "Die Werte werden korrigiert."
26            set SeiteA 0
27            set SeiteB 0
28      }
29          next
30    }
31
32    unexport WertePruefen
33  }
34
35  oo::class create Rechteck {
36    superclass Figur
37    mixin Filter
38
39    variable Flaeche
40    variable SeiteA
41    variable SeiteB
42
43    filter WertePruefen
44
45    constructor {tmpName tmpSeiteA tmpSeiteB} {
46      next $tmpName
47      set SeiteA $tmpSeiteA
48      set SeiteB $tmpSeiteB
49    }
50
51    method Flaeche {} {
52      set Flaeche [expr 1.0 * $SeiteA * $SeiteB]
53    }
54
55    export Flaeche
56  }
57
58  Rechteck create Rechteck1 "Rechteck1" 3 5
59  Rechteck1 Anzeigen
60  Rechteck1 Flaeche
61  Rechteck1 Anzeigen
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_353.tcl
Die Werte werden korrigiert.
Rechteck1: Fläche=unbekannt
Rechteck1: Fläche=0.0
oliver@debian:~$
```

In den Zeilen 19 bis 33 wird die Filter-Klasse `Filter` mit der Filter-Methode `WertePruefen` definiert. In Zeile 37 wird die Filter-Klasse in die Klasse `Rechteck` hineingemixt. In Zeile 43 wird die Filter-Methode festgelegt.

Listing 32.18: Filter importieren und Parameter weitergeben (Beispiel477.tcl)

```
1  #!/usr/bin/env tclsh
2
3  oo::class create Figur {
4    variable Name
5    variable Flaeche
6
7    constructor {tmpName} {
8      set Name $tmpName
9      set Flaeche "unbekannt"
10   }
11
12   method Anzeigen {} {
13     puts "$Name: Flaeche=$Flaeche"
14   }
15
16   export Anzeigen
17 }
18
19 oo::class create Filter {
20   variable SeiteA
21   variable SeiteB
22
23   method WertePruefen {args} {
24     if {$SeiteA < 0 || $SeiteB < 0} {
25           puts "Die Werte werden korrigiert."
26           set SeiteA 0
27           set SeiteB 0
28     }
29         return [next {*}$args]
30   }
31
32   unexport WertePruefen
33 }
34
35 oo::class create Rechteck {
36   superclass Figur
37   mixin Filter
38
39   variable Flaeche
```

```
40    variable SeiteA
41    variable SeiteB
42
43    filter WertePruefen
44
45    constructor {tmpName tmpSeiteA tmpSeiteB} {
46      next $tmpName
47      set SeiteA $tmpSeiteA
48      set SeiteB $tmpSeiteB
49    }
50
51    method Flaeche {Anteil} {
52      set Flaeche [expr 1.0 * $SeiteA * $SeiteB * $Anteil]
53    }
54
55    export Flaeche
56  }
57
58  Rechteck create Rechteck1 "Rechteck1" 3 5
59  Rechteck1 Anzeigen
60  Rechteck1 Flaeche 0.5
61  Rechteck1 Anzeigen
```

```
oli@debian: ~
oli@debian:~$ ./Beispiel477.tcl
Rechteck1: Flaeche=unbekannt
Rechteck1: Flaeche=7.5
oli@debian:~$
```

In diesem Beispiel wird jetzt der Methode `Flaeche` in Zeile 60 noch ein Parameter übergeben. Damit dieser Wert der Methode in Zeile 51 zur Verfügung steht, muss er von der Methode `WertePruefen` in Zeile 23 übernommen und an die Methode `Flaeche` weitergereicht werden. Deshalb wird in Zeile 23 die erste Klammer in der Form `{args}` geschrieben. In Zeile 29 wird mit dem Befehl `return [next {*}$args]` der Parameter an die folgende Methode `Flaeche` weitergegeben.

33 Grafische Benutzerschnittstelle (GUI)

Für die Darstellung grafischer Schnittstellen muss die erste Zeile des Programms geändert werden. Statt

```
#!/usr/bin/env tclsh
```

lautet die erste Zeile

```
#!/usr/bin/env wish
```

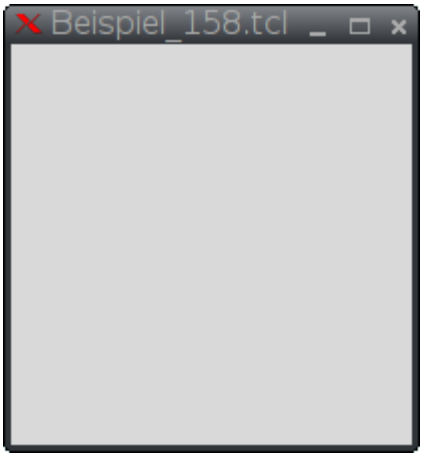
Die grafische Benutzerschnittstelle (GUI) besteht aus einem oder mehreren Fenstern, in das Elemente (z. B. Button, Textfelder) platziert werden und das vom Anwender mit der Maus bedient wird. Das Hauptfenster hat den Namen `.` (ein Punkt). Alle Elemente in dem Fenster sind dem Hauptfenster untergeordnet. So hat ein Rahmen im Hauptfenster z. B. den Namen `.rahmen` und ein Button im Rahmen hat z. B. den Namen `.rahmen.button`. Die Namen der Elemente müssen nach dem Punkt (für das Hauptfenster) immer mit einem Kleinbuchstaben beginnen. Wenn ein Element erzeugt wird, wird es noch nicht angezeigt. Zum Anzeigen des Elements wird ein Geometrie-Manager aufgerufen. Davon gibt es drei zur Auswahl: `pack`, `grid` und `place`.

Wenn man unter Windows einen Text oder Variableninhalt mit dem Befehl `puts` auf die Konsole ausgeben will, muss man durch das Tcl-Progamm erst eine Konsole starten (siehe Kapitel 34 auf Seite 327). Bei Linux besteht das Problem nicht.

Soll Ihr Programm auf verschiedenen Betriebssystemen eingesetzt werden, dann beachten Sie die Hinweise im Kapitel 23.2 auf Seite **??**.

Listing 33.1: Ein leeres Fenster (Beispiel158.tcl)

```
1 #!/usr/bin/env wish
```



Dieses sehr kurze Programm erzeugt ein Fenster ohne weiteren Inhalt. Zum Schließen muss man das Fenster erst mit der Maus ein wenig größer ziehen, damit die Fensterdekoration (z.B. das x zum Schließen des Fensters) sichtbar wird.

Listing 33.2: Ein Fenster mit einem Button (Beispiel159.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::button .meinButton -text "Schliessen" -command exit
4 pack .meinButton
```

Das Fenster wird in Zeile 1 erzeugt. In Zeile 3 wird ein Button erstellt, aber noch nicht angezeigt. Er hat den Namen `.meinButton` und ist ein Unterelement des Hauptfensters (das durch einen Punkt repräsentiert wird). In Zeile 4 wird der Button angezeigt.

34 Gleichzeitige Nutzung von GUI und Konsole (nur für Windows)

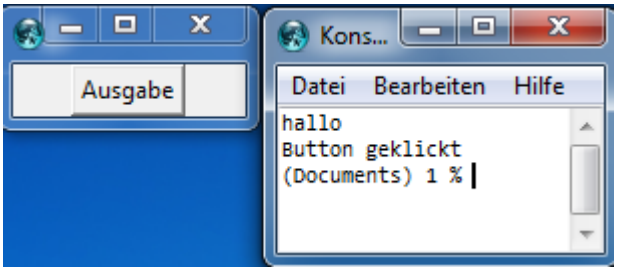
Befehl:

- `catch {console show}`

Wenn ein Programm mit GUI läuft, kann man unter Windows nicht mit dem Befehl `puts` einen Text oder einen Variableninhalt auf die Konsole ausgeben (unter Linux funktioniert das). Während des Programmierens oder einer Fehlersuche ist es aber oft hilfreich, mit `puts` eine Ausgabe zu erzeugen. Damit das auch unter Windows funktioniert gibt es den Befehl `console show`. Da der Befehl unter Linux eine Fehlermeldung auslöst, wird der Befehl am Besten in einen `catch`-Befehl geschrieben.

Listing 34.1: Ausgabe in eine Konsole unter Windows (Beispiel445.tcl)

```
#!/usr/bin/env wish

catch {console show}
puts hallo

ttk::button .bt -text "Ausgabe" -command {puts "Button geklickt"}
pack .bt
```



35 Window Manager

Befehl:

- `wm Option Fenstername Wert`

Mit dem Befehl `wm` legt man diverse Fenstereinstellungen fest. Hierzu gehören z. B. der Fenstertitel, die minimale oder maximale Größe oder die Angabe, ob die Fenstergröße mit der Maus geändert werden kann.

Tabelle 35.1: Die wichtigsten Optionen

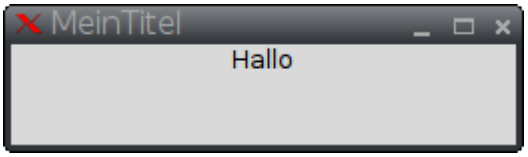
Option	Beschreibung
`wm title . "MeinTitel"`	Fenstertitel
`wm minsize . Breite Höhe`	Minimale Fenstergröße in Pixel
`wm maxsize . Breite Höhe`	Maximale Fenstergröße in Pixel
`wm resizable . Breite Höhe`	Legt fest, ob die Breite oder Höhe durch den Anwender veränderbar ist. Wird für Breite oder Höhe eine Null gesetzt, ist diese Dimension nicht veränderbar.
`geometry . BreitexHöhe +XPosition+YPosition`	Legt die Breite und Höhe des Fensters fest sowie die linke obere Ecke des Fensters. Die Breite und Höhe müssen nicht angegeben werden. Beispiel: `wm geometry . 150x80+100+50`
`geometry . +XPosition+YPosition`	Legt die linke obere Ecke des Fensters. Die Plus-Zeichen gehören zu dem Befehl. Beispiel: `wm geometry . +100+50`
`[winfo geometry .]`	Fenstergröße abfragen
`wm protocol . WM_DELETE_WINDOW {Anweisungen}`	Führt Befehle aus, wenn das Fenster geschlossen wird

35.1 Fenstertitel und Fenstergröße festlegen

Listing 35.1: Fenstertitel und Minimal-/Maximalgröße (Beispiel160.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::label .lb -text "Hallo"
```

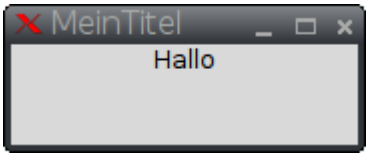
```
4 pack .lb
5 wm title . "MeinTitel"
6 wm minsize . 100 50
7 wm maxsize . 300 100
```



Ziehen Sie mit der Maus das Fenster größer und kleiner. Das Fenster kann nicht kleiner als die Mindestgröße und nicht größer als die Maximalgröße werden.

Listing 35.2: Fenster auf dem Bildschirm positionieren und eine Startgröße festlegen (Beispiel161.tcl)

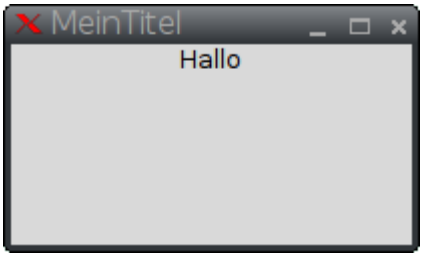
```
1 #!/usr/bin/env wish
2
3 ttk::label .lb -text "Hallo"
4 pack .lb
5 wm title . "MeinTitel"
6 wm geometry . 200x100+50+80
```



Das Fenster wird auf dem Bildschirm an x/y-Position 50/80 angezeigt. Die Startgröße beträgt 200 x 100 Pixel. Normalerweise macht man keine Größenangabe, sondern überlässt es dem Geometrie-Manager, die optimale Fenstergröße zu ermitteln.

Listing 35.3: Fenster mit unveränderbarer Größe (Beispiel162.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::label .lb -text "Hallo"
4 pack .lb
5 wm title . "MeinTitel"
6 wm geometry . 200x100+50+80
7 wm resizable . 0 0
```



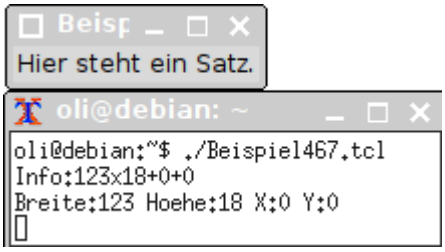
Sie können die Fenstergröße mit der Maus nicht ändern.

35.2 Fenstergröße abfragen

Mit dem Befehl `[winfo geometry .]` kann man die Größe eines Fensters ermitteln. Die Rückgabe ist in der Form `BreitexHoehe+X+Y`, wobei `X` und `Y` die x/y-Position bezeichnen. Diese sind für das Hauptfenster 0.

Listing 35.4: Fenstergröße abfragen (Beispiel467.tcl)

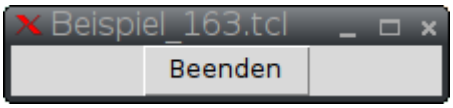
```
#!/usr/bin/env wish

catch {console show}

ttk::label .lbText -text "Hier steht ein Satz."
pack .lbText
update idletasks
set Info [winfo geometry .]
puts "Info:$Info"
set Liste [split $Info "x+"]
lassign $Liste Breite Hoehe X Y
puts "Breite:$Breite Hoehe:$Hoehe X:$X Y:$Y"
```



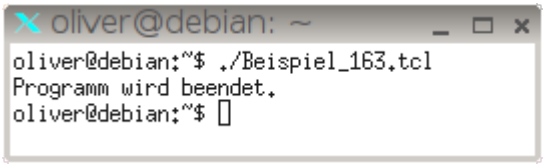
35.3 Fenster schließen / Programm beenden

Listing 35.5: Programm beenden (ohne wm protocol) (Beispiel163.tcl)

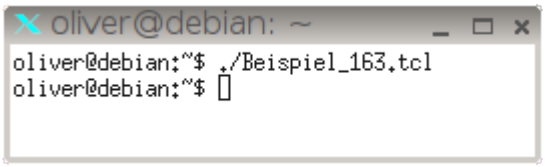
```
#!/usr/bin/env wish

proc ProgrammBeenden {} {
  # Befehle, die vor dem Programmende ausgefuehrt werden sollen
  puts "Programm wird beendet."
  destroy .
}

ttk::button .bt -text "Beenden" -command ProgrammBeenden
pack .bt
```



Wenn man auf den `Beenden`-Button klickt, sieht das Ergebnis wie folgt aus:



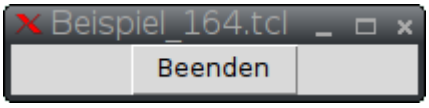
Wenn man das Fenster mit dem Kreuz schließt, sieht das Ergebnis so aus:



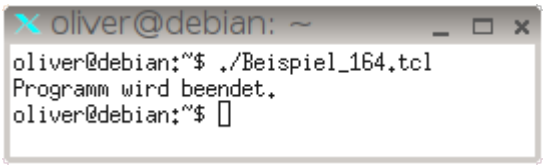
Die Prozedur `ProgrammBeenden` enthält alle Befehle, die direkt vor dem Schließen des Programms ausgeführt werden sollen. Allerdings wird die Prozedur nur aufgerufen, wenn der Anwender den Button `Beenden` anklickt. Wenn der Anwender das Programm über das Schließen-Kreuz rechts oben beendet, wird die Prozedur nicht ausgeführt. Um dennoch die Prozedur `ProgrammBeenden` auszuführen, muss man den Befehl `wm protocol . WM_DELETE_WINDOW` hinzufügen.

Listing 35.6: Programm beenden (mit wm protocol) (Beispiel164.tcl)

```
#!/usr/bin/env wish

proc ProgrammBeenden {} {
  # Befehle, die vor dem Programmende ausgefuehrt werden sollen
  puts "Programm wird beendet."
  destroy .
}

wm protocol . WM_DELETE_WINDOW {
  .bt invoke
}

ttk::button .bt -text "Beenden" -command ProgrammBeenden
pack .bt
```



Wenn man das Fenster mit dem Kreuz schließt, sieht das Ergebnis so aus:



In Zeile 9 wurde der Befehl `wm protocol` hinzugefügt. Der Befehl stellt sicher, dass beim Schließen des Fensters über das Schließen-Kreuz rechts oben der Button `Beenden` ausgeführt wird. Der Befehl `.bt invoke` in Zeile 10 simuliert einen Mausklick auf den `Beenden`-Button. Mehr dazu in Kapitel 47 auf Seite 549.

36 Überblick über die GUI-Elemente

Tcl/Tk hat zwei Arten von Elemente: die neueren Elemente beginnen mit `ttk::`, die klassischen Elemente sind ohne dieses Präfix. Wenn möglich, sollte man die neuen `ttk`-Elemente verwenden, weil man damit einfacher ein einheitliches Aussehen (Schrift, Farbe usw.) aller Elemente erreicht. Die einzelnen Elemente werden in Kapitel 41 auf Seite 389 ausführlich behandelt.

Tabelle 36.1: Grafische Elemente

Element	Beschreibung
`ttk::frame` `frame`	Rahmen
`ttk::labelframe` `labelframe`	Rahmen mit Titel
`ttk::scrollbar` `scrollbar`	Scrollbar (Laufleiste)
`ttk::label` `label`	Label (nicht editierbares Textfeld)
`ttk::button` `button`	Button (Schaltknopf)
`ttk::entry` `entry`	Eingabefeld
`ttk::checkbutton` `checkbutton`	Ankreuzkästchen
`ttk::radiobutton` `radiobutton`	Auswahlbutton
`ttk::menubutton` `manubutton`	Button mit Menüauswahl (aber auch ein Button innerhalb eines Menüs)
`ttk::spinbox` `spinbox`	Auswahlliste
`listbox` `(ttk::treeview)`	Auswahlliste
`ttk::combobox` `combobox`	Auswahlliste

Tabelle 36.1: Grafische Elemente

Element	Beschreibung
`ttk::scale` `scale`	Schieberegler
`ttk::panedwindow` `panedwindow`	Geteiltes Fenster
`ttk::notebook`	Fenster mit Reiter
`ttk::progressbar`	Fortschrittsanzeige
`text`	Textfeld zum Anzeigen und Bearbeiten von Text
`canvas`	Zeichenfläche
`menu`	Menü
`ttk::menubutton` `menubutton`	Menü-Button (aber auch ein Button außerhalb eines Menüs, der ein Menü anzeigt)
`ttk::treeview`	Ansicht in Baumstruktur
`widget::calendar`	Kalender
`toplevel`	(weiteres) Hauptfenster mit Fensterdekoration

Das klassische Element `Listbox` hat kein entsprechendes neues `ttk`-Element. Stattdessen verwendet man das Element `ttk::treeview`.

37 Geometrie-Manager

Die Geometrie-Manager platzieren die Elemente auf dem Bildschirm. Sie berechnen, wie viel Platz ein Element braucht und stellen jedes Element genau so groß dar, wie nötig. Wenn alle Elemente platziert sind, vergrößern bzw. verkleinern sie das Fenster, so dass es genau um die Elemente passt. Es gibt drei Geometrie-Manager: `pack`, `grid` und `place`.

37.1 Pack

Der Geometrie-Manager `pack` platziert der Reihe nach die Elemente auf dem Bildschirm (im Unterschied zum Geometrie-Manager `grid`, der die Elemente tabellarisch anordnet). Zum besseren Verständnis werden folgende Begriffe definiert:

- Element: das ist das darzustellende Element, z. B. ein Label, eine Scroll-Leiste usw.
- Parzelle: das ist eine rechteckige Fläche, die das Element aufnimmt.
- Leerraum: das ist die gesamte freie Fläche, die für die Parzelle zur Verfügung steht.

Als Veranschaulichung kann man sich eine Weide vorstellen, auf der jedes Schaf einen eigenen umzäunten Bereich hat (jedes Schaf steht somit alleine). Die Weide ist der Leerraum, jede Umzäunung bildet eine Parzelle und jedes Schaf ist ein Element.

Wenn der Geometrie-Manager ein Element zeichnet, berechnet er zunächst die Mindestgröße des Elements und schrumpft dann die übergeordnete Parzelle und den Leerraum soweit, dass das Element genau hineinpasst. Somit sieht man normalerweise keinen Unterschied zwischen der Größe des Elements und der Größe seiner Parzelle. Wenn die gesamt grafische Benutzeroberfläche nur aus genau einem Element besteht, schrumpft schließlich auch der Leerraum soweit, dass er genau so groß ist wie das Element. Erst wenn man mit der Maus das Programmfenster größer zieht, kann man die Unterschiede zwischen Element, Parzelle und Leerraum erkennen.

Wenn der Anwender das Fenster mit der Maus verkleinert, werden nacheinander die einzelnen Elemente ausgeblendet. Das zuletzt mit dem Befehl `pack` dargestellte Element verschwindet zuerst. Deshalb sollte man die wichtigsten Elemente (Menü, Scroll-Leisten, Buttons) zuerst erstellen, damit das Fenster bis zum Schluss bedienbar bleibt.

Tabelle 37.1: Die wichtigsten Optionen

Option	Beschreibung

`-side top`
`-side left`
`-side right`
`-side bottom`
	legt fest, wo die Parzelle im Leerraum des Fensters platziert werden soll
	top =oben
	left = links
	right = rechts
	bottom = unten

`-expand yes`	legt fest, ob die Parzelle vergrößert werden kann, so dass sie den gesamten Leerraum ausfüllt. Statt `yes` kann man auch `1` schreiben.

`-fill x`
`-fill y`
`-fill both`
	legt fest, ob und in welche Richtung das Element innerhalb der Parzelle vergrößert werden kann

`-anchor nw`	wenn das Element kleiner ist als die Parzelle, legt man mit dieser Option fest, wo das Element innerhalb der Parzelle platziert werden soll n = oben (north) e = rechts (east) s = unten (south) w = links (west)
`-ipadx Pixel`	horizontaler Abstand der Parzelle zum Rand des Leerraums (in Pixel)
`-ipady Pixel`	vertikaler Abstand der Parzelle zum Rand des Leerraums (in Pixel)
`-padx Pixel`	horizontaler Abstand des Elements zu seinem Parzellenrand (in Pixel)
`-pady Pixel`	vertikaler Abstand des Elements zu seinem Parzellenrand (in Pixel)

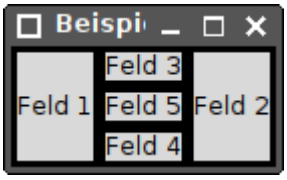
Im Folgenden werden der `pack`-Befehl und die Optionen näher betrachtet.
Die Option `-side` legt fest, wo die Parzelle im Leerraum eingefügt werden soll:

- -side top = oben
- -side left = links
- -side right = rechts
- -side bottom = unten

Immer wenn ein Element platziert wird, schrumpft der verbleibende Platz im Fenster, in den das nächste Element gesetzt wird. Wenn ein Element mit der Option oben angeordnet wird (`-side top`), verbleibt für das nächste Element der Platz darunter. Somit bestimmt die Reihenfolge, in der die Elemente platziert werden, das Aussehen der GUI.

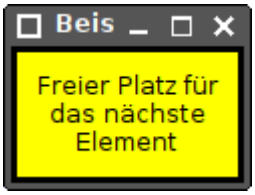
Listing 37.1: Elemente platzieren (Beispiel166.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::label .lb1 -text "Feld 1" -borderwidth 3 -relief ↩
       solid -anchor center
4  ttk::label .lb2 -text "Feld 2" -borderwidth 3 -relief ↩
       solid -anchor center
5  ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief ↩
       solid -anchor center
6  ttk::label .lb4 -text "Feld 4" -borderwidth 3 -relief ↩
       solid -anchor center
7  ttk::label .lb5 -text "Feld 5" -borderwidth 3 -relief ↩
       solid -anchor center
8
9  pack .lb1 -side left -expand yes -fill both
10 pack .lb2 -side right -expand yes -fill both
11 pack .lb3 -side top -expand yes -fill both
12 pack .lb4 -side bottom -expand yes -fill both
13 pack .lb5
```

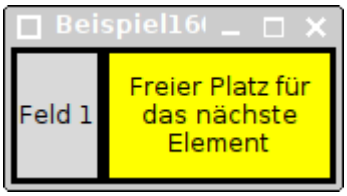


Nachstehend die Anordnung der Element Schritt für Schritt:

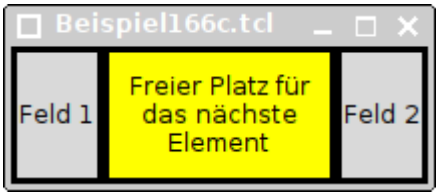
Vor Zeile 9:



Nach Zeile 9:

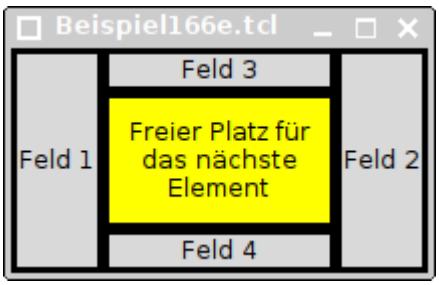


Nach Zeile 10:

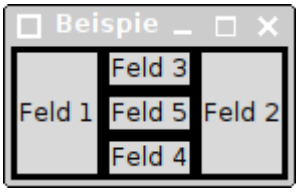


Nach Zeile 11:

Nach Zeile 12:



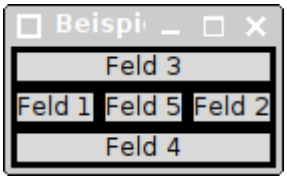
Nach Zeile 13:



Listing 37.2: Die Reihenfolge der Befehle bestimmt die Anordnung der Elemente (Beispiel167.tcl)

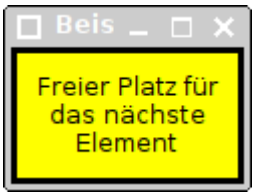
```
1 #!/usr/bin/env wish
2
3 ttk::label .lb1 -text "Feld 1" -borderwidth 3 -relief ↩
    solid -anchor center
4 ttk::label .lb2 -text "Feld 2" -borderwidth 3 -relief ↩
    solid -anchor center
5 ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief ↩
    solid -anchor center
6 ttk::label .lb4 -text "Feld 4" -borderwidth 3 -relief ↩
    solid -anchor center
```

```
7  ttk::label .lb5 -text "Feld 5" -borderwidth 3 -relief ↩
       solid -anchor center
8
9  pack .lb3 -side top -expand yes -fill both
10 pack .lb4 -side bottom -expand yes -fill both
11 pack .lb1 -side left -expand yes -fill both
12 pack .lb2 -side right -expand yes -fill both
13 pack .lb5
```

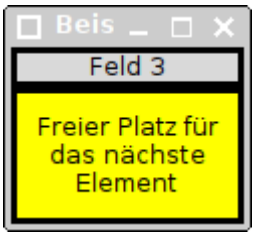


Nachstehend die Anordnung der Element Schritt für Schritt:

Vor Zeile 9:

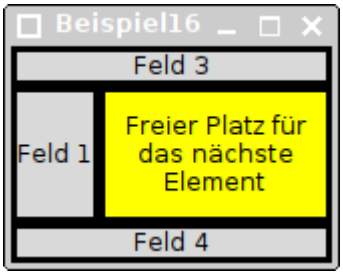


Nach Zeile 9:

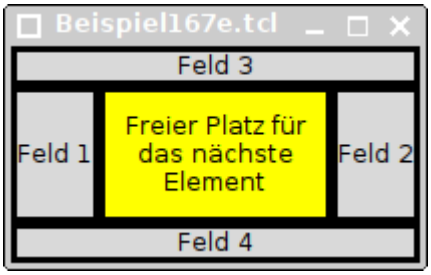


Nach Zeile 10:

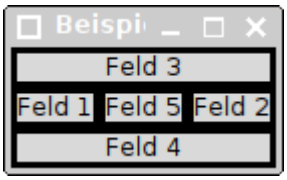
Nach Zeile 11:



Nach Zeile 12:

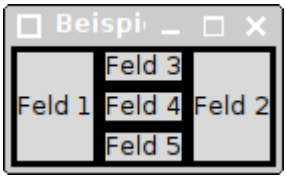


Nach Zeile 13:



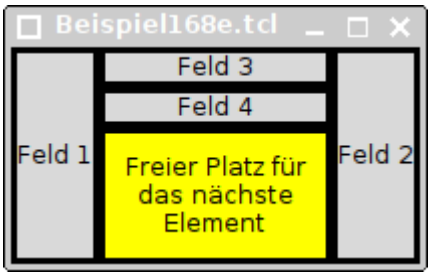
Listing 37.3: Feld D wird nicht unten, sondern oben platziert (Beispiel168.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::label .lb1 -text "Feld 1" -borderwidth 3 -relief ↩
      solid -anchor center
4  ttk::label .lb2 -text "Feld 2" -borderwidth 3 -relief ↩
      solid -anchor center
5  ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief ↩
      solid -anchor center
6  ttk::label .lb4 -text "Feld 4" -borderwidth 3 -relief ↩
      solid -anchor center
7  ttk::label .lb5 -text "Feld 5" -borderwidth 3 -relief ↩
      solid -anchor center
8
9  pack .lb1 -side left -expand yes -fill both
10 pack .lb2 -side right -expand yes -fill both
11 pack .lb3 -side top -expand yes -fill both
12 pack .lb4 -side top -expand yes -fill both
13 pack .lb5
```

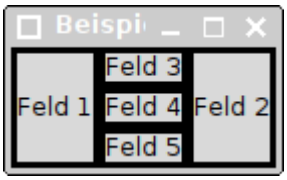


Nach Zeile 11:

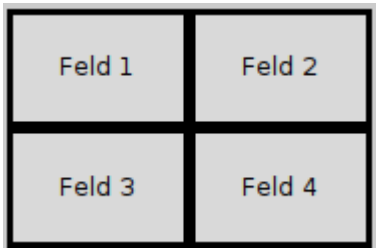
Nach Zeile 12:



Nach Zeile 13:



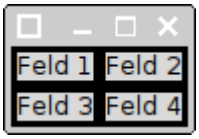
Wird ein Element links angeordnet, belegt das Element die gesamte Höhe des freien Platzes, so dass man kein weiteres Element darüber oder darunter anordnen kann. Wird ein Element oben angeordnet, so belegt es die gesamte Breite des freien Platzes, so dass man kein weiteres Element links oder rechts davon anordnet kann. Somit ist folgende Anordnung (zunächst) nicht möglich:



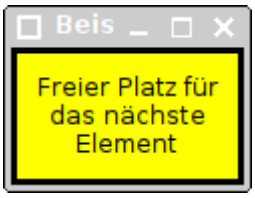
Erst wenn man Rahmen (Frames) einsetzt, kann man Elemente beliebig anordnen. In einem Rahmen kann man die Elemente wie in einem eigenen Fenster anordnen.

Listing 37.4: Rahmen (Frames) (Beispiel169.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::frame .fr1
4  ttk::frame .fr2
5
6  pack .fr1 -side top -expand yes -fill both
7  pack .fr2 -side bottom -expand yes -fill both
8
9  ttk::label .fr1.lb1 -text "Feld 1" -borderwidth 3 -relief ↩
       solid -anchor center
10 ttk::label .fr1.lb2 -text "Feld 2" -borderwidth 3 -relief ↩
       solid -anchor center
11 ttk::label .fr2.lb3 -text "Feld 3" -borderwidth 3 -relief ↩
       solid -anchor center
12 ttk::label .fr2.lb4 -text "Feld 4" -borderwidth 3 -relief ↩
       solid -anchor center
13
14 pack .fr1.lb1 -side left -expand yes -fill both
15 pack .fr1.lb2 -side right -expand yes -fill both
16 pack .fr2.lb3 -side left -expand yes -fill both
17 pack .fr2.lb4 -side right -expand yes -fill both
```

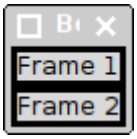


Vor Zeile 6:

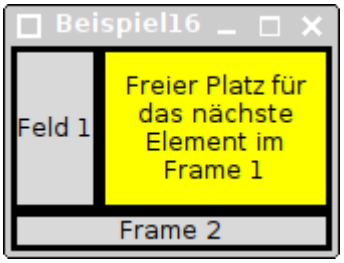


Nach Zeile 6:

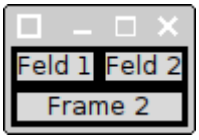
Nach Zeile 7:



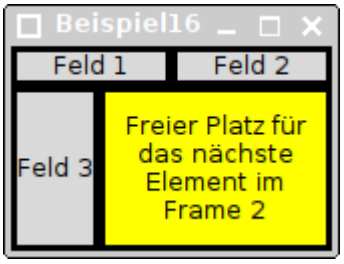
Nach Zeile 14:



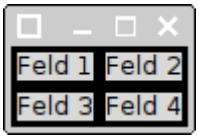
Nach Zeile 15:



Nach Zeile 16:



Nach Zeile 17:



Nun zu den unterschiedlichen Möglichkeiten, die Elemente anzuordnen. Hierzu gibt es folgende Optionen:

Tabelle 37.2: Die wichtigsten Optionen

Option	Beschreibung

`-side left`
`-side right`
`-side top`
`-side bottom`
	Platziert die Parzelle im Fenster links, rechts, oben oder unten

`-padx Pixel` `-pady Pixel`	Abstand vom Element zum Rand der Parzelle (in Pixel)

`-anchor n`
`-anchor ne`
`-anchor e`
`-anchor se`
`-anchor s`
`-anchor sw`
`-anchor w`
`-anchor nw`
`-anchor center`
	Verankerung des Elements in seiner Parzelle (Himmelsrichtung n=Nord, e=Ost, s=Süd, w=West). Die Verankerung kann auch an zwei Seiten erfolgen (z. B. `-anchor ne` oder `-anchor nw`). Die Verankerung ist nur wirksam und sichtbar, wenn die Parzelle größer ist als das Element.

`-fill x`
`-fill y`
`-fill both`
	Vergrößert das Element in seiner Parzelle, so dass es genau so groß ist, wie die Parzelle.

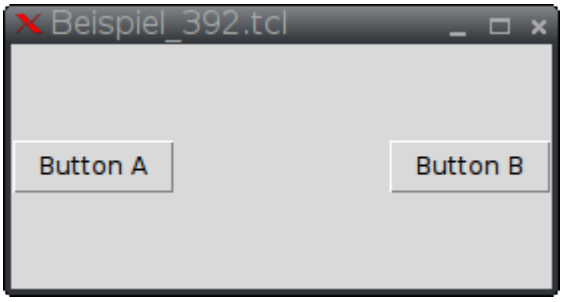
`-expand yes`	Vergrößert die Parzelle, wenn sich die Fenstergröße ändert.

Die Optionen werden an Hand der folgenden Beispiele vorgestellt.

Listing 37.5: Platzierung mit -side (Beispiel392.tcl)

```
#!/usr/bin/env wish

ttk::button .bt1 -text "Button A"
ttk::button .bt2 -text "Button B"

pack .bt1 -side left
pack .bt2 -side right
```

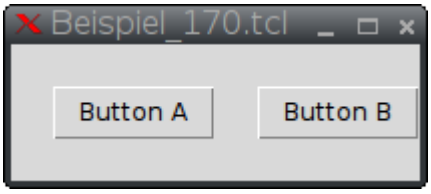
Wenn man das Fenster mit der Maus vergrößert:



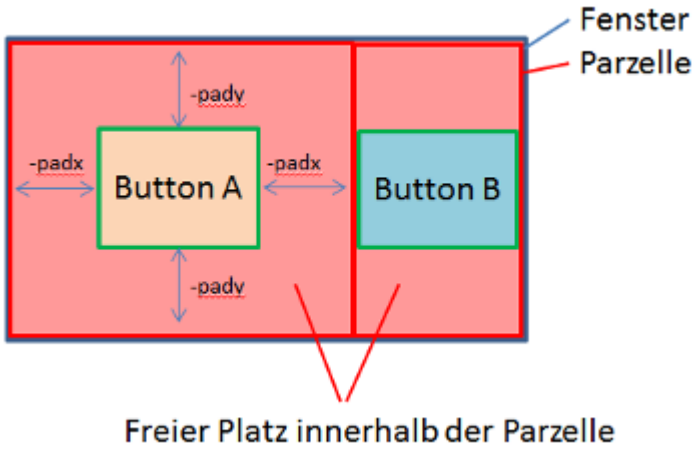
Wenn das Fenster erscheint, wird die Parzelle für `Button A` (bzw. `Button B`) soweit geschrumpft, dass die Parzelle exakt die Größe des Buttons hat. Danach wird das Fenster ebenfalls soweit verkleinert, bis es genau um die beiden Parzellen passt. Zieht man das Fenster größer, entsteht Platz zwischen den beiden Parzellen. Die Parzellen sind weiterhin genau so groß wie der Button. Die beiden Parzellen werden im Fenster mit den Optionen `-side left` bzw. `-side right` in der Fensterfläsche platziert.

Listing 37.6: Abstand des Elements zum Rand der Parzelle mit -padx und -pady (Beispiel170.tcl)

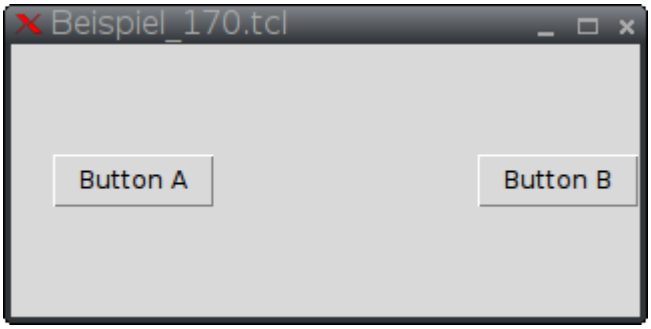
```
#!/usr/bin/env wish

ttk::button .bt1 -text "Button A"
ttk::button .bt2 -text "Button B"

pack .bt1 -side left -padx 20 -pady 20
pack .bt2 -side right
```



Die Optionen `-padx` und `-pady` legen den Abstand des Elements zum Rand der Parzelle fest. Somit ist die Parzelle um den `Button A` jetzt größer als der Button. In Folge dessen ist auch das Fenster größer geworden. Außerdem wird dadurch automatisch auch die Parzelle um den `Button B` größer (höher). Der `Button B` wird zentriert in seiner Parzelle angezeigt. Die nachstehende Abbildung verdeutlicht das:



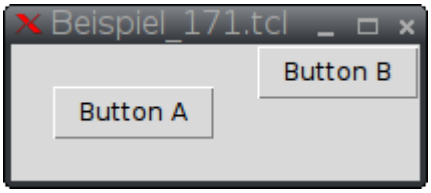
Zieht man das Fenster größer bleibt die Parzelle des `Buttons A` an der linken Fensterseite und die Parzelle des `Buttons B` an der rechten Seite.



Man erkennt außerdem, dass die Parzellen nicht größer geworden sind. Denn die Buttons, die zentriert in ihrer Parzelle angezeigt werden, haben ihren Abstand zum linken bzw. rechten Fensterrand beibehalten.

Listing 37.7: Ausrichtung des Elements innerhalb seiner Parzelle mit -anchor (Beispiel171.tcl)

```
#!/usr/bin/env wish

ttk::button .bt1 -text "Button A"
ttk::button .bt2 -text "Button B"

pack .bt1 -side left -padx 20 -pady 20
pack .bt2 -side right -anchor n
```



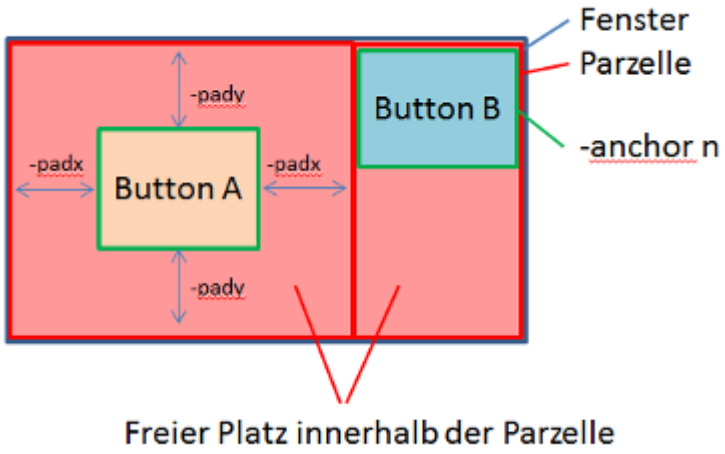
Durch die Option `-pady` vergrößert sich die Parzelle von `Button A`. Dadurch wird automatisch auch die Parzelle von `Button B` höher. Mit der Option `-anchor` legt man deshalb die Seite bzw. Ecke fest, wo das Element innerhalb seiner Parzelle erscheinen soll. Die Option `-anchor` kennt folgende (Himmels-)Richtungen:

n / ne / e / se / s / sw / w / nw / center

gleichbedeutend mit

Nord / Nordost / Ost / Südost / Süd / Südwest / West / Nordwest / Mitte.

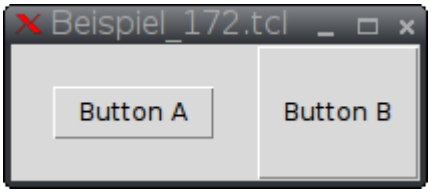
Die nachstehende Abbildung zeigt, dass der `Button B` oben (im Norden) angeordnet wird:



Man kann auch bestimmen, dass das Element größer wird, wenn mehr Platz in der Parzelle zur Verfügung steht. Dies geschieht mit der Option `-fill`.

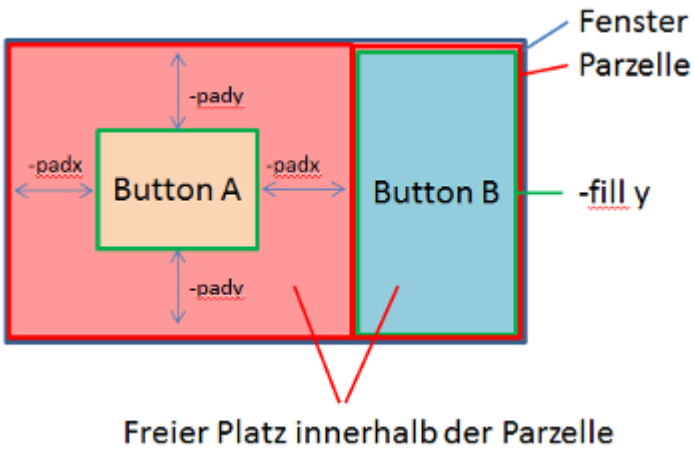
Listing 37.8: Vergrößern des Elements innerhalb seiner Parzelle mit -fill (Beispiel172.tcl)

```
#!/usr/bin/env wish

ttk::button .bt1 -text "Button A"
ttk::button .bt2 -text "Button B"

pack .bt1 -side left -padx 20 -pady 20
pack .bt2 -side right -fill y
```



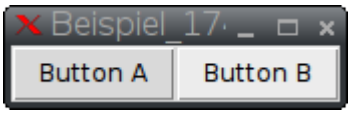
Die Option `-fill` vergrößert das Element innerhalb seiner Parzelle. Die Option `-fill x`

vergrößert waagrecht, die Option `-fill y` senkrecht und `-fill both` vergrößert in beide Richtungen. Der `Button B` wurde durch die Option `-fill y` senkrecht vergrößert, so dass er jetzt die gesamte Parzelle ausfüllt.

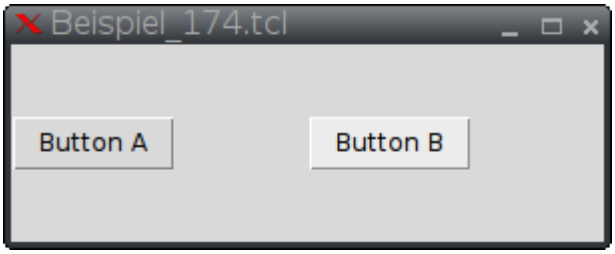


Listing 37.9: Vergrößern der Parzelle mit -expand yes (Beispiel174.tcl)

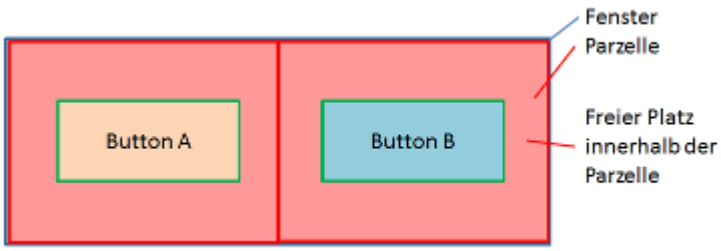
```
#!/usr/bin/env wish

ttk::button .bt1 -text "Button A"
ttk::button .bt2 -text "Button B"

pack .bt1 -side left
pack .bt2 -side right -expand yes
```



Wenn man das Fenster mit der Maus vergrößert:

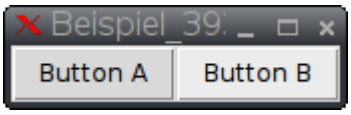


Mit der Option `-expand yes` vergrößert man die Parzelle, wenn das Fenster mit der Maus größer gezogen wird. Das Element wird dabei in seiner Größe nicht verändert. Das kann man am `Button B` erkennen. Seine Parzelle hat sich der größeren Fenstergröße angepasst, aber er wird weiterhin in seiner ursprünglichen Größe zentriert angezeigt.

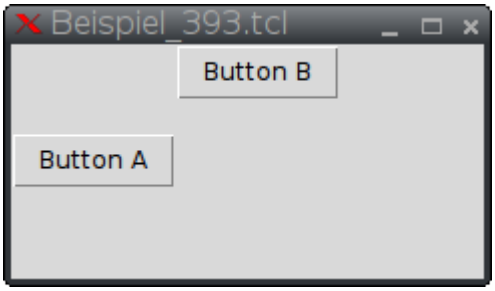


Listing 37.10: Vergrößern der Parzelle mit -expand yes und platzieren des Elements mit -anchor (Beispiel393.tcl)

```
#!/usr/bin/env wish

ttk::button .bt1 -text "Button A"
ttk::button .bt2 -text "Button B"

pack .bt1 -side left
pack .bt2 -side right -expand yes -anchor nw
```



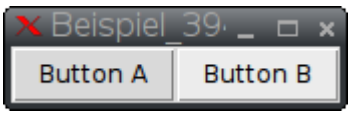
Wenn man das Fenster mit der Maus vergrößert:



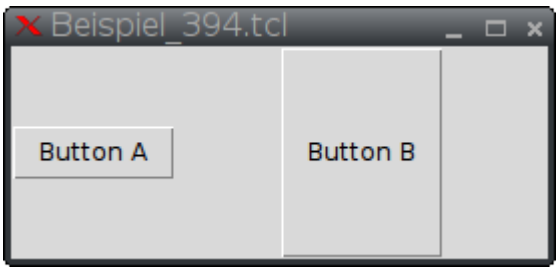
Durch die Option `-expand yes` wird die Parzelle größer. Der `Button B` wird durch die Option `-anchor nw` links oben in der vergrößerten Parzelle platziert.

Listing 37.11: Vergrößern des Fensters mit -expand yes und -fill y (Beispiel394.tcl)

```
#!/usr/bin/env wish

ttk::button .bt1 -text "Button A"
ttk::button .bt2 -text "Button B"

pack .bt1 -side left
pack .bt2 -side right -expand yes -fill y
```



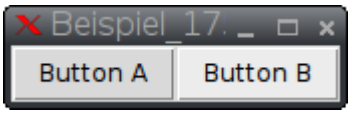
Wenn man das Fenster mit der Maus vergrößert:



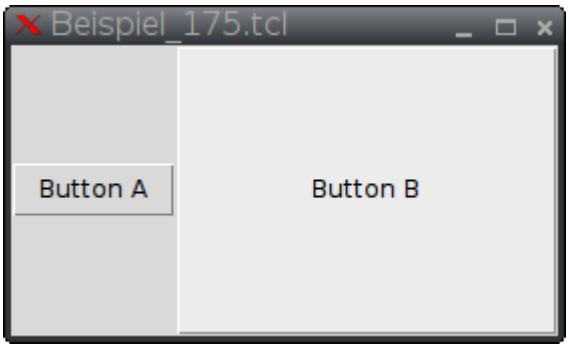
Durch die Option `-expand yes` wird die Parzelle des `Buttons B` größer, wenn das Fenster vergrößert wird. Innerhalb der vergrößerten Parzelle kann sich dann der `Button B` durch die Option `-fill y` senkrecht ausdehnen.

Listing 37.12: Vergrößern des Fensters mit -expand yes und -fill both (Beispiel175.tcl)

```
#!/usr/bin/env wish

ttk::button .bt1 -text "Button A"
ttk::button .bt2 -text "Button B"

pack .bt1 -side left
pack .bt2 -side right -expand yes -fill both
```



Wenn man das Fenster mit der Maus vergrößert:



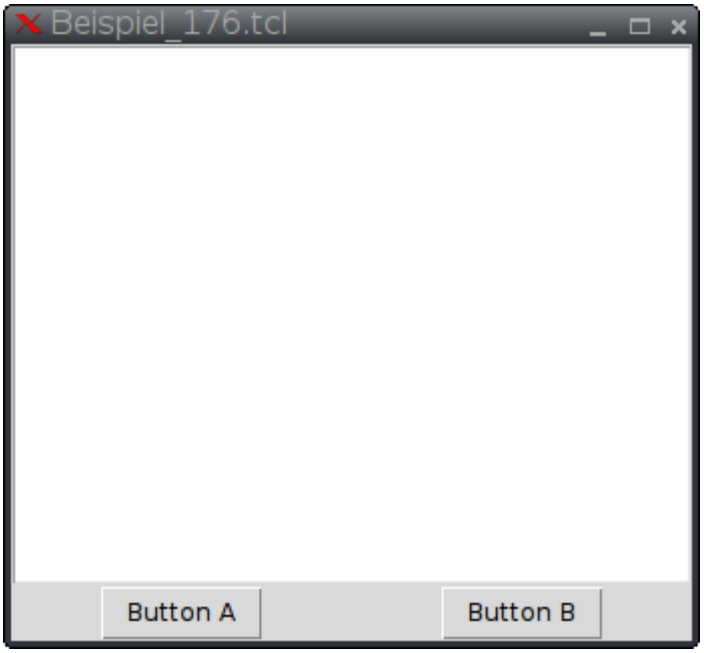
Durch die Option `-expand yes` wird die Parzelle des `Buttons B` größer, wenn das Fenster vergrößert wird. Durch die Option `-fill both` wird dann der Button sowohl waagrecht als auch senkrecht bis an den Rand der Parzelle ausgedehnt.

In der Regel bekommen nur wenige Elemente die Option `-expand yes` zugewiesen. Denn Buttons, Scroll-Leisten, das Menü usw. sollen normalerweise ihre Größe nicht ändern, wenn das Fenster vergrößert wird.

Listing 37.13: Nur die Listbox wird größer, wenn man das Fenster mit der Maus vergrößert (Beispiel176.tcl)

```
#!/usr/bin/env wish

listbox .lbox
ttk::button .bt1 -text "Button A"
ttk::button .bt2 -text "Button B"

pack .lbox -side top -expand yes -fill both
pack .bt1 -side left -expand yes
pack .bt2 -side right -expand yes
```

Wenn man das Fenster mit der Maus vergrößert:



Die Option `-expand yes` bei den Buttons in den Zeilen 8 und 9 hat den Sinn, die Buttons horizontal gleichmäßig auszurichten, wenn das Fenster vergrößert wird.

37.2 Grid

Der Geometriemanager `grid` ordnet die Elemente wie in einem unsichtbaren Gitter zeilen- und spaltenweise an. Die erste Zeile hat den Index 0, die erste Spalte den Index 0.

Tabelle 37.3: Die wichtigsten Optionen

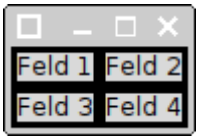
Option	Beschreibung
`-row Zeile`	Platziert das Element in eine bestimmte Zeile
`-column Spalte`	Platziert das Element in eine bestimmte Spalte
`-rowspan Anzahl`	Legt das Element über mehrere Zeilen
`-columnspan Anzahl`	Legt das Element über mehrere Spalten
`-in`	Legt ein Element in ein anderes Element hinein
`-padx Pixel`	Horizontaler Abstand des Elements zu seinem Zellenrand (in Pixel)
`-pady Pixel`	Vertikaler Abstand des Elements zu seinem Zellenrand (in Pixel)

Tabelle 37.3: Die wichtigsten Optionen

Option	Beschreibung
`-sticky nesw`	Klebt den Rand des Elements an einen oder mehrere Zellenränder an n = oben (north) e = rechts (east) s = unten (south) w = links (west)
`grid rowconfigure Element Zeile -weight Zahl`	Gewichtet die Zeile beim Vergrößern des Fensters. `weight 0` bedeutet, die Zeile wird nicht vergrößert.
`grid columnconfigure Element Spalte -weight Zahl`	Gewichtet die Spalte beim Vergrößern des Fensters. `weight 0` bedeutet, die Spalte wird nicht vergrößert.

Listing 37.14: Vier Elemente im Quadrat (Beispiel177.tcl)

```
#!/usr/bin/env wish

ttk::label .lb1 -text "Feld 1" -borderwidth 3 -relief solid
ttk::label .lb2 -text "Feld 2" -borderwidth 3 -relief solid
ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief solid
ttk::label .lb4 -text "Feld 4" -borderwidth 3 -relief solid

grid .lb1 -row 0 -column 0
grid .lb2 -row 0 -column 1
grid .lb3 -row 1 -column 0
grid .lb4 -row 1 -column 1
```



Listing 37.15: Ein Element über mehrere Spalten (Beispiel178.tcl)

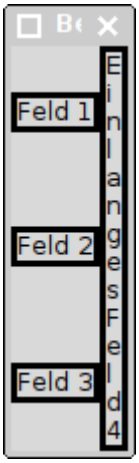
```
#!/usr/bin/env wish

```

```
3  ttk::label .lb1 -text "Feld 1" -borderwidth 3 -relief ↩
     solid
4  ttk::label .lb2 -text "Feld 2" -borderwidth 3 -relief ↩
     solid
5  ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief ↩
     solid
6  ttk::label .lb4 -text "Ein langes Feld 4" -borderwidth 3 ↩
     -relief solid
7
8  grid .lb1 -row 0 -column 0
9  grid .lb2 -row 0 -column 1
10 grid .lb3 -row 0 -column 2
11 grid .lb4 -row 1 -column 0 -columnspan 3
```

In Zeile 11 wird das Label über drei Spalten gespannt.

Listing 37.16: Ein Element über mehrere Zeilen (Beispiel179.tcl)

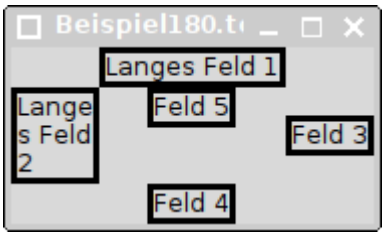
```
1  #!/usr/bin/env wish
2
3  ttk::label .lb1 -text "Feld 1" -borderwidth 3 -relief ↩
     solid
4  ttk::label .lb2 -text "Feld 2" -borderwidth 3 -relief ↩
     solid
5  ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief ↩
     solid
6  ttk::label .lb4 -text "Ein langes Feld 4" -borderwidth 3 ↩
     -relief solid -wraplength 1
7
8  grid .lb1 -row 0 -column 0
9  grid .lb2 -row 1 -column 0
10 grid .lb3 -row 2 -column 0
11 grid .lb4 -row 0 -column 1 -rowspan 3
```



In Zeile 11 wird das Label über drei Zeilen gespannt. In Zeile 6 wird die maximale Breite des Labels auf ein Zeichen begrenzt. Sonst würde sich der Text nicht vertikal ausdehnen (Anmerkung: man kann beim Label den Text nicht um z. B. 90 Grad drehen).

Listing 37.17: Das mittlere Element oben ankleben (Beispiel180.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::label .lb1 -text "Langes Feld 1" -borderwidth 3 ↩
     -relief solid
4  ttk::label .lb2 -text "Langes Feld 2" -borderwidth 3 ↩
     -relief solid -wraplength 40
5  ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief ↩
     solid
6  ttk::label .lb4 -text "Feld 4" -borderwidth 3 -relief ↩
     solid
7  ttk::label .lb5 -text "Feld 5" -borderwidth 3 -relief ↩
     solid
8
9  grid .lb1 -row 0 -column 1
10 grid .lb2 -row 1 -column 0
11 grid .lb3 -row 1 -column 2
12 grid .lb4 -row 2 -column 1
13 grid .lb5 -row 1 -column 1 -sticky n
```

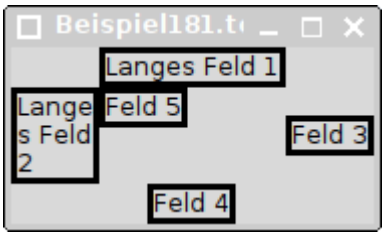


In Zeile 13 wird das Label am oberen Rand (n=North) verankert.

Listing 37.18: Das mittlere Element links oben ankleben (Beispiel181.tcl)

```
1  #!/usr/bin/env wish
```

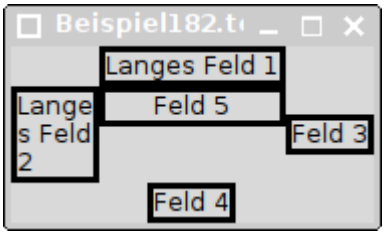
```
2
3 ttk::label .lb1 -text "Langes Feld 1" -borderwidth 3 ↩
    -relief solid
4 ttk::label .lb2 -text "Langes Feld 2" -borderwidth 3 ↩
    -relief solid -wraplength 40
5 ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief ↩
    solid
6 ttk::label .lb4 -text "Feld 4" -borderwidth 3 -relief ↩
    solid
7 ttk::label .lb5 -text "Feld 5" -borderwidth 3 -relief ↩
    solid
8
9 grid .lb1 -row 0 -column 1
10 grid .lb2 -row 1 -column 0
11 grid .lb3 -row 1 -column 2
12 grid .lb4 -row 2 -column 1
13 grid .lb5 -row 1 -column 1 -sticky nw
```



In Zeile 13 wird das Label an der Ecke links oben (nw=North West) verankert.

Listing 37.19: Das mittlere Element oben ankleben (über die gesamte Zellenbreite) (Beispiel182.tcl)

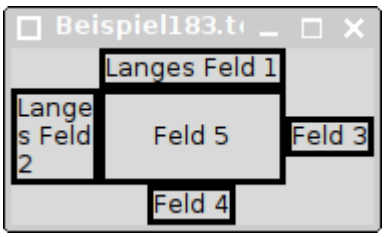
```
1 #!/usr/bin/env wish
2
3 ttk::label .lb1 -text "Langes Feld 1" -borderwidth 3 ↩
    -relief solid
4 ttk::label .lb2 -text "Langes Feld 2" -borderwidth 3 ↩
    -relief solid -wraplength 40
5 ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief ↩
    solid
6 ttk::label .lb4 -text "Feld 4" -borderwidth 3 -relief ↩
    solid
7 ttk::label .lb5 -text "Feld 5" -borderwidth 3 -relief ↩
    solid -anchor center
8
9 grid .lb1 -row 0 -column 1
10 grid .lb2 -row 1 -column 0
11 grid .lb3 -row 1 -column 2
12 grid .lb4 -row 2 -column 1
13 grid .lb5 -row 1 -column 1 -sticky new
```



In Zeile 13 wird das Label am oberen Rand von ganz links bis ganz rechts (new=North East West) verankert.

Listing 37.20: Das mittlere Element an allen Seiten ankleben (es füllt somit die gesamte Zelle) (Beispiel183.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::label .lb1 -text "Langes Feld 1" -borderwidth 3 ↩
     -relief solid
4  ttk::label .lb2 -text "Langes Feld 2" -borderwidth 3 ↩
     -relief solid -wraplength 40
5  ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief ↩
     solid
6  ttk::label .lb4 -text "Feld 4" -borderwidth 3 -relief ↩
     solid
7  ttk::label .lb5 -text "Feld 5" -borderwidth 3 -relief ↩
     solid -anchor center
8
9  grid .lb1 -row 0 -column 1
10 grid .lb2 -row 1 -column 0
11 grid .lb3 -row 1 -column 2
12 grid .lb4 -row 2 -column 1
13 grid .lb5 -row 1 -column 1 -sticky nesw
```



In Zeile 13 wird das Label von links oben bis rechts unten (nesw=North East South West) verankert.
Vergrößert man das Fenster mit der Maus, werden die Zellen nicht automatisch vergrößert. Mit den Befehlen

```
grid columnconfigure . Spalte -weight Zahl
grid rowconfigure . Zeile -weight Zahl
```

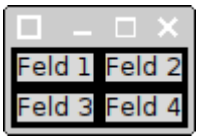
kann man die Zeilen und Spalten vergrößern. Die Zahl hinter `-weight` gibt die Gewichtung an:

Tabelle 37.4: Die Option weight

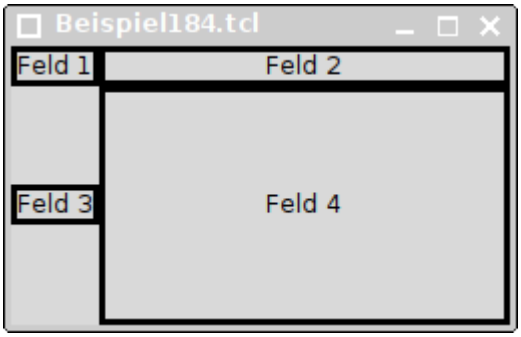
Option	Beschreibung
`-weight 0`	die Spalte oder Zeile wird nicht vergrößert (das ist die Vorbelegung)
`-weight 1`	die Spalte oder Zeile wird vergrößert
`-weight 2`	die Spalte oder Zeile wird doppelt so stark vergrößert wie eine Spalte oder Zeile mit dem Wert `-weight 1`.

Listing 37.21: Spalten und Zeilen werden unterschiedlich vergrößert (Beispiel184.tcl)

```
#!/usr/bin/env wish

ttk::label .lb1 -text "Feld 1" -borderwidth 3 -relief solid
ttk::label .lb2 -text "Feld 2" -borderwidth 3 -relief solid -anchor center
ttk::label .lb3 -text "Feld 3" -borderwidth 3 -relief solid
ttk::label .lb4 -text "Feld 4" -borderwidth 3 -relief solid -anchor center

grid .lb1 -row 0 -column 0
grid .lb2 -row 0 -column 1 -sticky nesw
grid .lb3 -row 1 -column 0
grid .lb4 -row 1 -column 1 -sticky nesw

grid columnconfigure . 1 -weight 1
grid rowconfigure . 1 -weight 1
```



Vergrößert man das Fenster mit der Maus, bleiben die erste Spalte und die erste Zeile unverändert. Aber die zweite Spalte und zweite Zeile werden vergrößert.



37.3 Mischen der Geometriemanager

Man kann die Geometrie-Manager auch mischen, also gleichzeitig einsetzen.

Listing 37.22: grid und pack gleichzeitig verwenden (Beispiel185.tcl)

```
1  #!/usr/bin/env wish
2
3  listbox .lbox
4  ttk::frame .fr
5
6  ttk::label .fr.lb1 -text "Feld 1"
7  ttk::label .fr.lb2 -text "Feld 2"
8  ttk::label .fr.lb3 -text "Feld 3"
9  ttk::label .fr.lb4 -text "Feld 4"
10
11 grid .fr.lb1 -row 0 -column 0
12 grid .fr.lb2 -row 0 -column 1
13 grid .fr.lb3 -row 1 -column 0
14 grid .fr.lb4 -row 1 -column 1
15
16 pack .lbox -side left
17 pack .fr -side right -anchor n
```

In den Zeilen 3 und 4 werden eine Listbox und ein Rahmen definiert, die in den Zeilen 16 und 17 mit dem Befehl `pack` angeordnet werden. In den Zeilen 6 bis 9 werden vier Labels definiert, die mit dem Befehl `grid` innerhalb des Rahmens angeordnet werden.

37.4 Place

Der Geometrie-Manager `place` wird normalerweise nicht gebraucht. Er ist der flexibelste Geometrie-Manager, aber auch der komplizierteste. Für weitere Infos sollte man im Internet nachlesen.

38 Scroll-Leisten

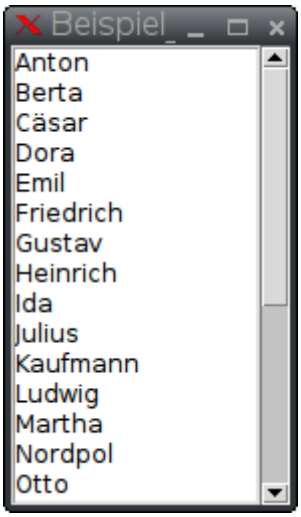
Einige Elemente können mit Scroll-Leisten kombiniert werden. Damit beim Verkleinern des Fensters durch den Anwender die Scroll-Leisten nicht vor dem zugehörigen Element verschwinden, platziert man zuerst die Scroll-Leisten und danach das zugehörige Element. Es ist oft von Vorteil, das Element und die zugehörigen Scroll-Leisten in einem Rahmen (`frame`) zusammenzufassen, damit sie gemeinsam platziert werden können.

38.1 Scroll-Leisten mit pack

Im Folgenden wird gezeigt, wie man Scroll-Leisten mit dem Geometriemanager `pack` verwendet.

Listing 38.1: Senkrechte Scroll-Leiste (Beispiel186.tcl)

```
#!/usr/bin/env wish

set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav Heinrich Ida Julius Kaufmann}
set Liste [concat $Liste {Ludwig Martha Nordpol Otto Paula Quelle Richard Samuel Schule Theodor}]
set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe Ypsilon Zacharias}]

listbox .lbox -width 15 -height 15 -yscrollcommand {.sbY set} -listvariable Liste
ttk::scrollbar .sbY -command {.lbox yview}

pack .sbY -side right -fill y
pack .lbox -side left
```



In Zeile 7 wird die Listbox durch die Option `-yscrollcommand {.sbY set}` mit der Scroll-Leiste `.sbY` verknüpft. In Zeile 8 wird die Scroll-Leiste `.sbY` erstellt und mit dem Befehl `-command {.lbox yview}` angewiesen, den sichtbaren Teil der Listbox gemäß der Position des Schiebers in der Scroll-Leiste anzupassen.

Listing 38.2: Waagrechte Scroll-Leiste (Beispiel187.tcl)

```
1 #!/usr/bin/env wish
2
3 set Liste {Donaudampfschiffahrt Telekommunikationsunternehmen}
4
5 listbox .lbox -width 15 -height 15 -xscrollcommand {.sbX set} -listvariable Liste
6 ttk::scrollbar .sbX -orient horizontal -command {.lbox xview}
7
8 pack .sbX -side bottom -fill x
9 pack .lbox -side top
```

Für eine waagrechte Scroll-Leiste muss die Orientierung der Scroll-Leiste auf horizontal eingestellt werden. Das erfolgt mit der Option `-orient horizontal` in Zeile 6. Standardmäßig ist die Scroll-Leiste vertikal ausgerichtet.

Listing 38.3: Waagrechte und senkrechte Scroll-Leiste (ohne Rahmen) (Beispiel188.tcl)

```
1 #!/usr/bin/env wish
2
3 set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav Heinrich Ida Julius Kaufmann}
4 lappend Liste {Ludwig Martha Nordpol Otto Paula Quelle Richard Samuel Schule Theodor}
5 set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe Ypsilon Zacharias}]
6
```

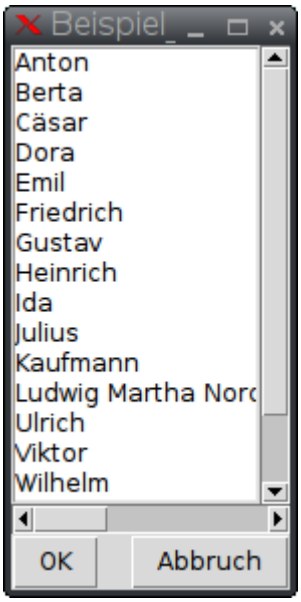
```
listbox .lbox -width 15 -height 15 -xscrollcommand {.sbX set} -yscrollcommand {.sbY set} -listvariable Liste
ttk::scrollbar .sbX -orient horizontal -command {.lbox xview}
ttk::scrollbar .sbY -command {.lbox yview}

pack .sbX -side bottom -fill x
pack .sbY -side right -fill y
pack .lbox -side top
```

In der Regel ist es vorteilhaft das Element und die Scroll-Leisten in einem Rahmen zusammenzufassen. Schließlich bilden die drei Elemente eine Einheit und sollten nicht auseinandergerissen werden.

Listing 38.4: Waagrechte und senkrechte Scroll-Leiste (mit Rahmen) (Beispiel189.tcl)

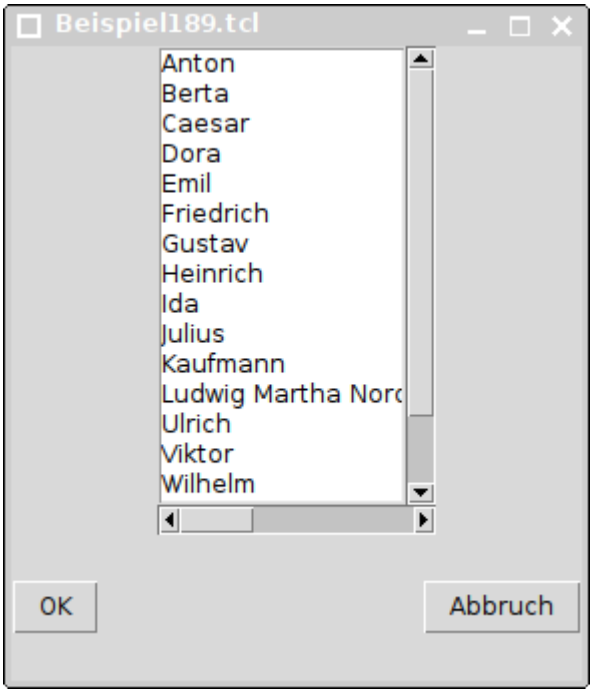
```
#!/usr/bin/env wish

set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav Heinrich Ida Julius Kaufmann}
lappend Liste {Ludwig Martha Nordpol Otto Paula Quelle Richard Samuel Schule Theodor}
set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe Ypsilon Zacharias}]

ttk::frame .fr
listbox .fr.lbox -width 15 -height 15 -xscrollcommand {.fr.sbX set} -yscrollcommand {.fr.sbY set} -listvariable Liste
ttk::scrollbar .fr.sbX -orient horizontal -command {.fr.lbox xview}
ttk::scrollbar .fr.sbY -command {.fr.lbox yview}

pack .fr.sbX -side bottom -fill x
pack .fr.sbY -side right -fill y
```

```
14 pack .fr.lbox -side top
15
16 ttk::button .btOK -text OK
17 ttk::button .btAbbruch -text Abbruch
18
19 pack .fr -side top
20 pack .btOK -side left
21 pack .btAbbruch -side right
```



Die Zeilen 7 bis 14 erstellen einen Rahmen, in den die Listbox sowie die beiden Scroll-Leisten platziert werden. In den Zeilen 19 bis 21 werden der gesamte Rahmen sowie die weiteren Elemente der Anwendung platziert.

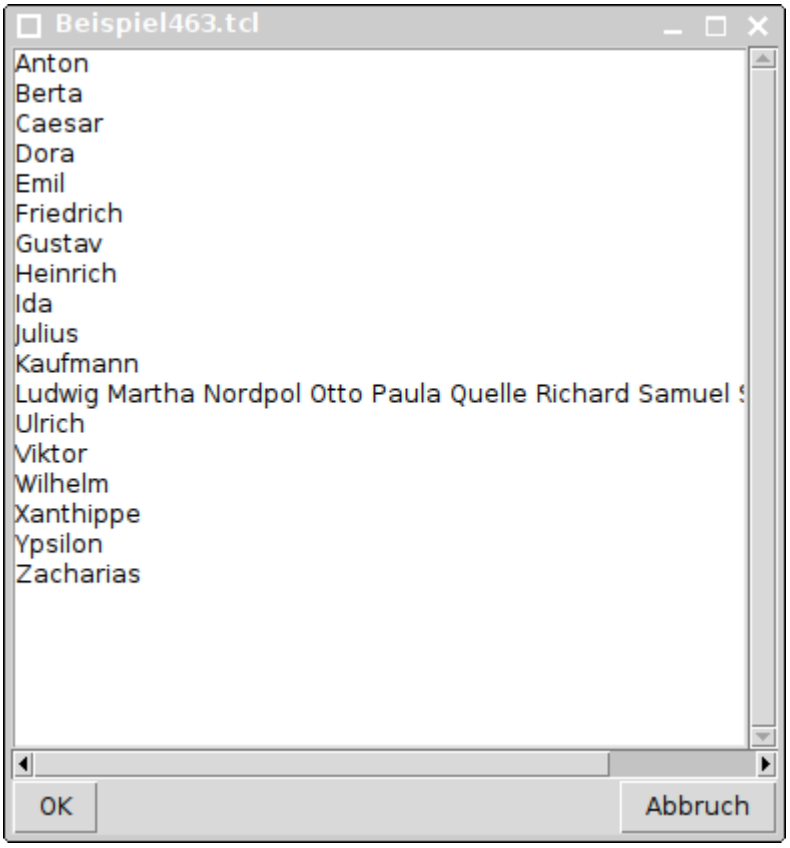
Im vorangegangenen Beispiel haben Sie vielleicht festgestellt, dass sich die Listbox nicht vergrößert, wenn Sie das Fenster mit der Maus größer ziehen.



Im folgende Beispiel passt sich die Listbox automatisch an die Rahmengröße an.

Listing 38.5: Listbox passt sich an die Rahmengröße an (Beispiel463.tcl)

```
#!/usr/bin/env wish

set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav Heinrich Ida Julius Kaufmann}
lappend Liste {Ludwig Martha Nordpol Otto Paula Quelle Richard Samuel Schule Theodor}
set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe Ypsilon Zacharias}]

ttk::frame .fr
listbox .fr.lbox -width 15 -height 15 -xscrollcommand {.fr.sbX set} -yscrollcommand {.fr.sbY set} -listvariable Liste
ttk::scrollbar .fr.sbX -orient horizontal -command {.fr.lbox xview}
ttk::scrollbar .fr.sbY -command {.fr.lbox yview}

pack .fr.sbX -side bottom -fill x
pack .fr.sbY -side right -fill y
pack .fr.lbox -side top -expand yes -fill both

ttk::button .btOK -text OK
ttk::button .btAbbruch -text Abbruch

pack .fr -side top -expand yes -fill both
pack .btOK -side left
pack .btAbbruch -side right
```



In den Zeilen 14 und 19 wurden die Optionen `-expand yes` und `-fill both` hinzugefügt. Beides zusammen bewirkt, dass sowohl der Rahmen `.fr` als auch die Listbox `.fr.lbox` automatisch ihre Größe anpassen. Beide Optionen wurden in Kapitel 37 auf Seite 337 genauer erklärt.

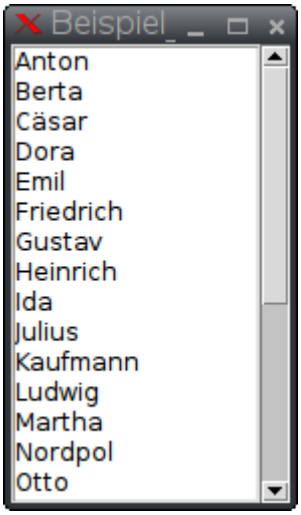
38.2 Scroll-Leisten mit grid

Im Folgenden wird gezeigt, wie man Scroll-Leisten mit dem Geometriemanager `grid` verwendet.

Listing 38.6: Senkrechte Scroll-Leiste (Beispiel190.tcl)

```
1 #!/usr/bin/env wish
2
3 set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav ↩
    Heinrich Ida Julius Kaufmann}
4 set Liste [concat $Liste {Ludwig Martha Nordpol Otto Paula↩
     Quelle Richard Samuel Schule Theodor}]
5 set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe ↩
    Ypsilon Zacharias}]
6
7 listbox .lbox -width 15 -height 15 -yscrollcommand {.sbY ↩
    set} -listvariable Liste
8 ttk::scrollbar .sbY -command {.lbox yview}
9
```

```
10 grid .sbY -row 0 -column 1 -sticky ns
11 grid .lbox -row 0 -column 0 -sticky nsew
12 grid rowconfigure . 0 -weight 1
13 grid columnconfigure . 0 -weight 1
```



In der Zeile 10 wird die Scroll-Leiste oben und unten angeklebt. In den Zeilen 12 und 13 wird festgelegt, dass die Listbox größer wird, wenn man das Fenster vergrößert.

Listing 38.7: Waagrechte Scroll-Leiste (Beispiel191.tcl)

```
1  #!/usr/bin/env wish
2
3  set Liste {Donaudampfschiffahrt ↩
      Telekommunikationsunternehmen}
4
5  listbox .lbox -width 15 -height 15 -xscrollcommand {.sbX ↩
      set} -listvariable Liste
6  ttk::scrollbar .sbX -orient horizontal -command {.lbox ↩
      xview}
7
8  grid .sbX -row 1 -column 0 -sticky we
9  grid .lbox -row 0 -column 0 -sticky nsew
10 grid rowconfigure . 0 -weight 1
11 grid columnconfigure . 0 -weight 1
```

In der Zeile 8 wird die Scroll-Leiste links und rechts angeklebt.

Listing 38.8: Waagrechte und senkrechte Scroll-Leiste (Beispiel192.tcl)

```
1  #!/usr/bin/env wish
2
3  set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav
      Heinrich Ida Julius Kaufmann}
4  lappend Liste {Ludwig Martha Nordpol Otto Paula Quelle
      Richard Samuel Schule Theodor}
5  set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe
      Ypsilon Zacharias}]
6
7  listbox .lbox -width 15 -height 15 -xscrollcommand {.sbX
      set} -yscrollcommand {.sbY set} -listvariable Liste
8  ttk::scrollbar .sbX -orient horizontal -command {.lbox
      xview}
9  ttk::scrollbar .sbY -command {.lbox yview}
10
11 grid .sbX -row 1 -column 0 -sticky we
12 grid .sbY -row 0 -column 1 -sticky ns
13 grid .lbox -row 0 -column 0 -sticky nsew
14 grid rowconfigure . 0 -weight 1
15 grid columnconfigure . 0 -weight 1
```

Beispiel
Anton
Berta
Cäsar
Dora
Emil
Friedrich
Gustav
Heinrich
Ida
Julius
Kaufmann
Ludwig Martha Norc
Ulrich
Viktor
Wilhelm

39 Bildschirmausgabe aktualisieren

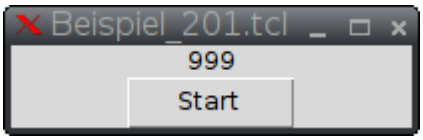
Befehle:

- `update`
- `update idletasks`

Normalerweise wartet Tcl/Tk mit der Aktualisierung der Bildschirmausgabe solange, bis das Programm im Leerlauf ist. Dies erfolgt aus Effizienzgründen. Wenn man eine sehr lang dauernde Berechnung durchführt, wird somit während der Berechnung die grafische Oberfläche nicht aktualisiert. Mit dem Befehl `update` bzw. `update idletasks` zwingt man die Anwendung die Bildschirmausgabe zu aktualisieren. Der Befehl `update` aktualisiert die gesamte Bildschirmausgabe. Der Befehl `update idletasks` aktualisiert nur die wichtigsten Elemente (das ist normalerweise ausreichend).

Listing 39.1: Bildschirmausgabe wird nicht aktualisiert (Beispiel201.tcl)

```
#!/usr/bin/env wish

proc Zaehlen {} {
  global Text
  for {set Zahl 0} {$Zahl < 1000} {incr Zahl} {
    set Text $Zahl
  }
}

set Text ""

ttk::label .lb -textvariable Text
ttk::button .bt -text "Start" -command Zaehlen

pack .lb -side top
pack .bt -side bottom
```



Wenn man den Button `Start` klickt, wird von 1 bis 999 gezählt. In der Prozedur `Zaehlen` wird die mit dem Label verknüpfte Textvariable `Text` zwar bei jedem Durchlauf hochgesetzt, aber es erfolgt keine Ausgabe auf dem Bildschirm. Der Bildschirm wird erst aktualisiert, wenn die Prozedur beendet und das Programm wieder im Leerlauf ist.

Listing 39.2: Bildschirmausgabe wird aktualisiert (Beispiel202.tcl)

```
#!/usr/bin/env wish
```

```
2
3 proc Zaehlen {} {
4   global Text
5   for {set Zahl 0} {$Zahl < 1000} {incr Zahl} {
6     set Text $Zahl
7     update idletasks
8   }
9 }
10
11 set Text ""
12
13 ttk::label .lb -textvariable Text
14 ttk::button .bt -text "Start" -command Zaehlen
15
16 pack .lb -side top
17 pack .bt -side bottom
```

Durch den Befehl `update idletasks` in Zeile 7 aktualisiert die Anwendung die Bildschirmausgabe. Allerdings wird dadurch die Ausführung des Programms etwas langsamer.

40 Vorgefertigte Dialoge

Es gibt folgende vorgefertigte Dialoge, die man aufrufen kann:

Tabelle 40.1: Vorgefertigte Dialoge

Dialog	Beschreibung
`tk_messageBox`	Ein Fenster mit einer Nachricht (z.B. eine Warnmeldung oder eine Ja-Nein-Abfrage) und vorgegebenen Buttons (z.B. OK, Yes, No, Cancel)
`tk_dialog`	Ein Fenster mit einer Nachricht und beliebigen Buttons
`tk_getOpenFile`	Datei öffnen Dialog
`tk_getSaveFile`	Datei speichern Dialog
`tk_chooseDirectory`	Auswählen eines Ordners
`tk_chooseColor`	Farbe auswählen

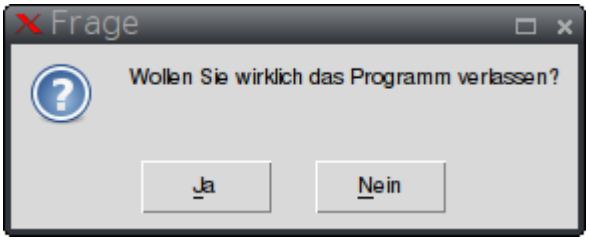
Das Aussehen dieser Dialoge (Schriftart, Schriftgröße usw.) legt man wie folgt fest:

- `font configure TkDefaultFont -family Helvetica -size 8 -slant roman -weight normal`
- `font configure TkTextFont -family Helvetica -size 8 -slant roman -weight normal`
- `font configure TkMenuFont -family Helvetica -size 8 -slant roman -weight normal`
- `font configure TkCaptionFont -family Helvetica -size 8 -slant roman -weight normal`

Listing 40.1: Schriftart und Schriftgröße festlegen (Beispiel205.tcl)

```
#!/usr/bin/env wish

font configure TkDefaultFont -family Helvetica -size 8 -slant roman -weight normal
font configure TkTextFont -family Helvetica -size 8 -slant roman -weight normal
font configure TkMenuFont -family Helvetica -size 8 -slant roman -weight normal
```

```
6 font configure TkCaptionFont -family Helvetica -size 8 ↩
    -slant roman -weight normal
7
8 tk_messageBox -title "Frage" -message "Wollen Sie wirklich↩
     das Programm verlassen?" -icon question -type yesno ↩
    -default no
```



40.1 Nachrichten-Fenster

Der Befehl `tk_messageBox` erzeugt ein Fenster, das eine Nachricht anzeigt und auf eine Antwort des Anwenders wartet.

Tabelle 40.2: Die wichtigsten Optionen

Option	Beschreibung
`-title "Titel"`	Titel
`-message "Text"`	Text

`-icon error`
`-icon info`
`-icon question`
`-icon warning`
	Bild

`-type abortretryignore`
`-type ok`
`-type okcancel`
`-type retrycancel`
`-type yesno`
`-type yesnocancel`
	Buttons

`-default abort`
`-default retry`
`-default ignore`
`-default ok`
`-default cancel`
`-default yes`
`-default no`
	Default-Button, der ausgeführt wird, wenn der Anwender (statt mit der Maus) nur die Return-Taste drückt.

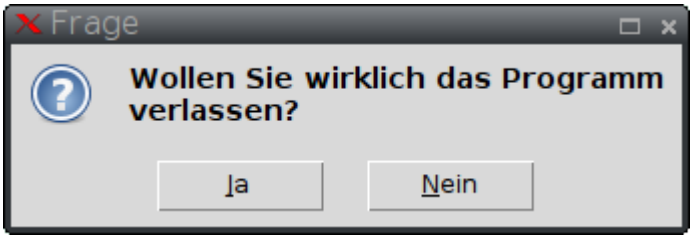
Tabelle 40.2: Die wichtigsten Optionen

Option	Beschreibung
`-parent Fenstername`	Name des Fensters, aus dem die Messagebox aufgerufen wird.

Die Option `-parent` sorgt dafür, dass nach dem Schließen der Messagebox das aufrufende Fenster wieder im Vordergrund erscheint.

Listing 40.2: Nachrichten-Fenster (Beispiel206.tcl)

```
1 #!/usr/bin/env wish
2
3 set Antwort [tk_messageBox -title "Frage" -message "Wollen
    Sie wirklich das Programm verlassen?" -icon question
   -type yesno -default no]
4 if {$Antwort == yes} {
5    exit
6 }
```



In Zeile 3 wird ein Nachrichtenfenster mit zwei Buttons erzeugt. Das Programm wartet solange, bis der Dialog vom Anwender geschlossen wird. Der vom Anwender angeklickte Button wird in der Variablen `Antwort` gespeichert und in Zeile 4 ausgewertet.

40.2 Dialog-Fenster

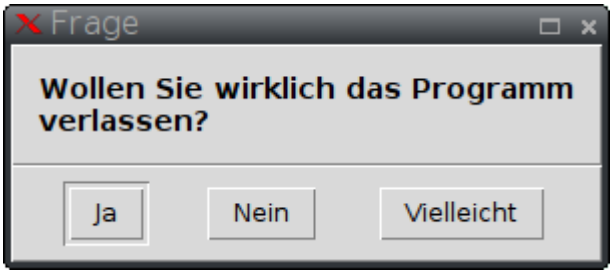
Der Befehl `tk_dialog` erzeugt ein Fenster, das eine Nachricht anzeigt und auf eine Antwort des Anwenders wartet. Im Unterschied zu `tk_messageBox` kann man beliebige Buttons und beliebige Bilder (Bitmaps) verwenden. Es gibt keine Optionen, sondern die Einstellungen des Dialogs erfolgen in einer festen Reihenfolge:

```
tk_dialog Elementname Titel Text Bitmap DefaultButton Button1 Button2 ...
```

Listing 40.3: Dialog-Fenster (Beispiel207.tcl)

```
#!/usr/bin/env wish

set Antwort [tk_dialog .meinDialog "Frage" "Wollen Sie
   wirklich das Programm verlassen?" "" 0 "Ja" "Nein" "
   Vielleicht"]
switch $Antwort {
  0 exit
```

```
  2 {tk_messageBox -message "Dann ueberlegen Sie noch mal.
    " -type ok}
}
```



Jeder Button bekommt einen Index zugeordnet. Der erste Button hat den Index 0. Die Angabe des Default-Buttons erfolgt über den Index, in dem Beispiel `0`, also der erste Button. Wenn kein Bitmap angezeigt werden soll, dann muss man einen leeren String `""` hinschreiben.

40.3 Datei öffnen-Dialog

Der Befehl `tk_getOpenFile` zeigt einen Dialog zum Öffnen einer Datei. Der Anwender kann nur eine bereits bestehende Datei auswählen. Er kann mit dem Befehl nicht eine Datei neu anlegen. Der Rückgabewert ist ein Dateiname bzw. eine Liste mit mehreren Dateinamen. Wenn der Dialog abgebrochen wird, wird ein leerer String zurückgegeben. Standardmäßig werden auch die versteckten Dateien und Ordner angezeigt. Mit Hilfe eines kleinen Tricks (siehe Beispiel) kann man die versteckten Dateien und Ordner auch ausblenden.

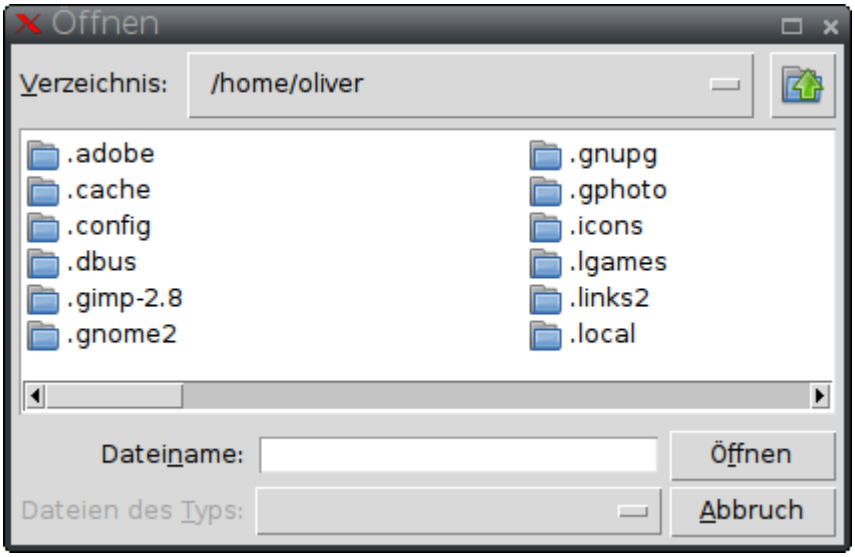
Tabelle 40.3: Die wichtigsten Optionen

Option	Beschreibung
`-title "Titel"`	Titel
`-filetypes Liste`	Dateitypen, die man öffnen kann
`-initialdir Ordner`	Ordner, der beim Öffnen des Dialogs bereits eingestellt ist
`-multiple yes`	erlaubt, mehrere Dateien gleichzeitig auszuwählen

Listing 40.4: Datei öffnen (Beispiel208.tcl)

```
#!/usr/bin/env wish

set Dateiname [tk_getOpenFile]

if {$Dateiname != ""} {
```

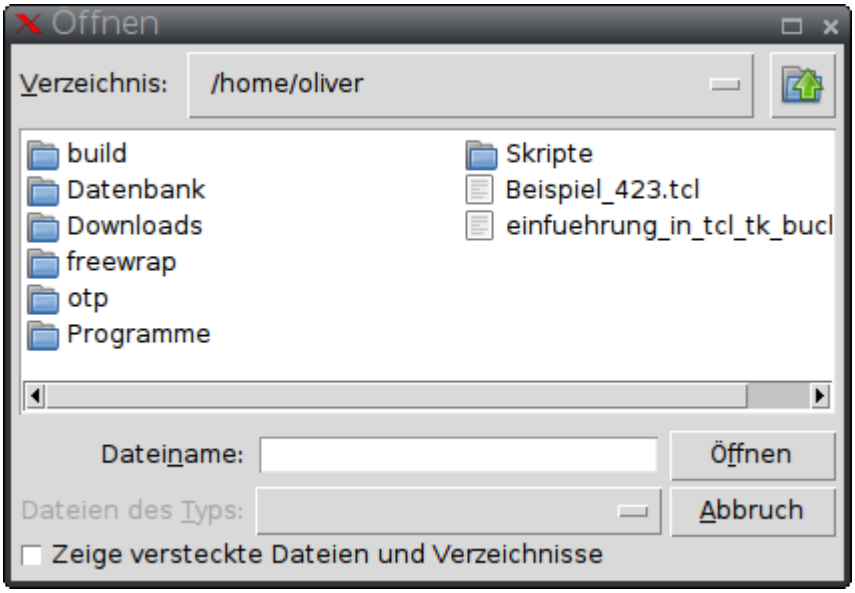
```
6   puts "$Dateiname wird geoeffnet."
7 }
```



Das tatsächliche Öffnen der Datei muss man noch programmieren (Zeile 6). Der Dialog gibt nur den Dateinamen zurück, welche Datei geöffnet werden soll.

Listing 40.5: Datei öffnen (versteckte Ordner und Dateien sind ausgeblendet) (Beispiel423.tcl)

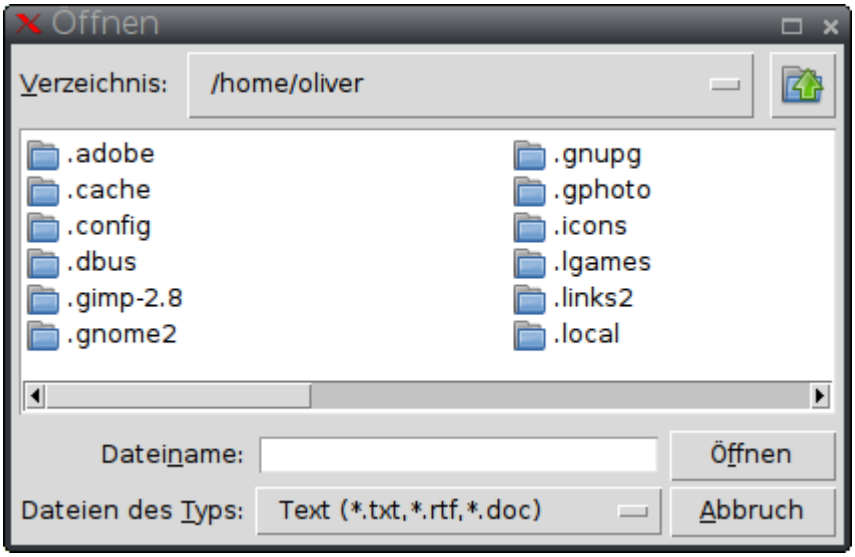
```
1  #!/usr/bin/env wish
2
3  catch {tk_getOpenFile foo bar}
4  set ::tk::dialog::file::showHiddenVar 0
5  set ::tk::dialog::file::showHiddenBtn 1
6
7  set Dateiname [tk_getOpenFile]
8
9  if {$Dateiname != ""} {
10   puts "$Dateiname wird geoeffnet."
11 }
```



Der Dialog `tk_getOpenFile` bietet standardmäßig keine Option, um versteckte Ordner und Dateien auszublenden. Deshalb ist folgender Trick notwendig: In Zeile 3 wird der Dialog aufgerufen und sofort wieder geschlossen, ohne dass der Anwender den Dialog sieht. Dadurch werden alle mit dem Dialog im Hintergrund verbundenen Variablen initialisiert, so dass sie in den beiden folgenden Zeilen geändert werden können. In Zeile 4 wird festgelegt, dass die versteckten Ordner und Dateien ausgeblendet werden sollen. In Zeile 5 wird eingestellt, dass der Anwender eine Option angezeigt bekommt, um die versteckten Ordner und Dateien einzublenden.

Listing 40.6: Voreingestellte Dateitypen (Beispiel209.tcl)

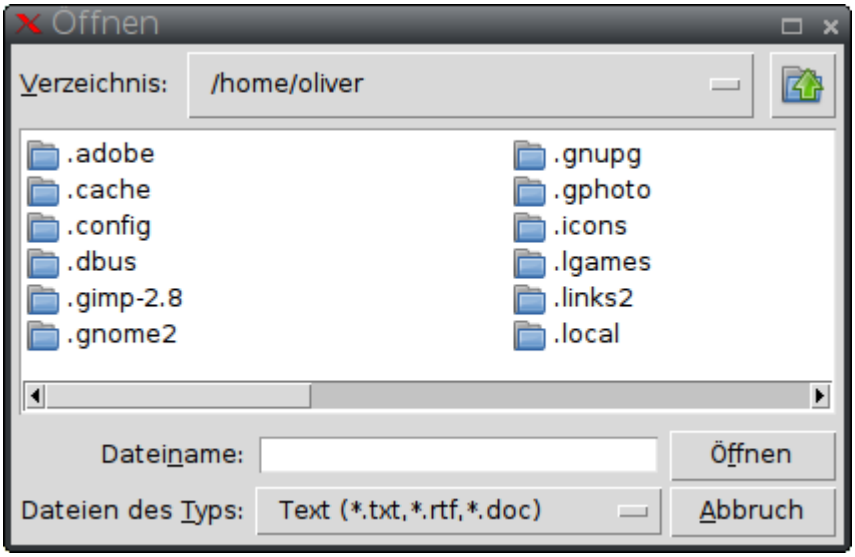
```
1 #!/usr/bin/env wish
2
3 set Dateiname [tk_getOpenFile -filetypes {{{Text} {*.txt
    *.rtf *.doc}} {{Alle} {*}}}]
4
5 if {$Dateiname != ""} {
6   puts "$Dateiname wird geoeffnet."
7 }
```



Durch die Option -filetypes wird die angezeigte Dateimenge auf ausgewählte Dateitypen beschränkt. Der Aufbau ist wie folgt:
`{{Bezeichnung} {Muster}} {{Bezeichnung} {Muster}}`.

Listing 40.7: Voreingestellte Dateitypen (Beispiel210.tcl)

```
#!/usr/bin/env wish

set Dateitypen {
  {{Text} {*.txt *.rtf *.doc}}
  {{Alle} {*}}
}
set Dateiname [tk_getOpenFile -filetypes $Dateitypen]

if {$Dateiname != ""} {
  puts "$Dateiname wird geoeffnet."
}
```

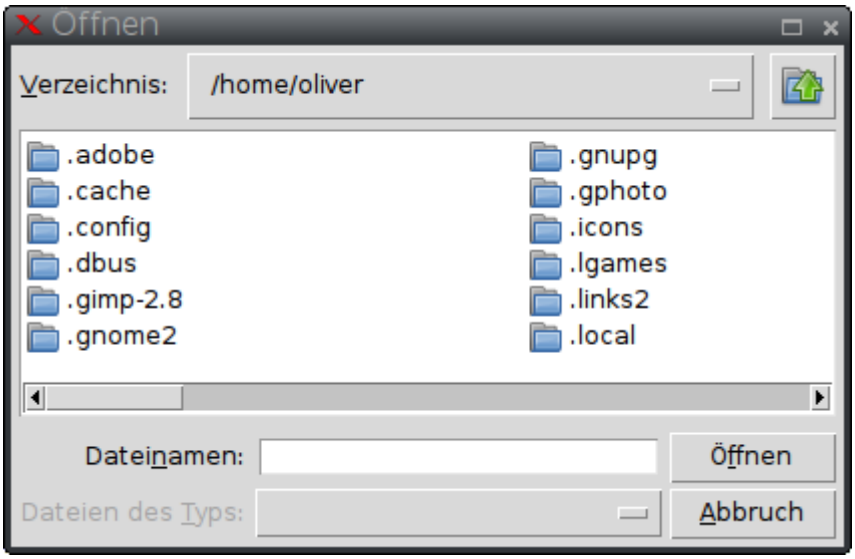


In den Zeilen 3 bis 6 werden die Dateitypen definiert. Der Aufbau ist wie folgt:

```
{{Bezeichnung} {Muster}}
{{Bezeichnung} {Muster}}
```

Listing 40.8: Mehrere Dateien auswählen (mit der Strg-Taste) (Beispiel211.tcl)

```
1 #!/usr/bin/env wish
2
3 set Dateiname [tk_getOpenFile -multiple yes]
4
5 if {$Dateiname != ""} {
6   foreach Datei $Dateiname {
7     puts "$Datei wird geoeffnet."
8   }
9 }
```



Die Rückgabe mehrerer ausgewählter Dateinamen erfolgt als Liste.

40.4 Datei speichern-Dialog

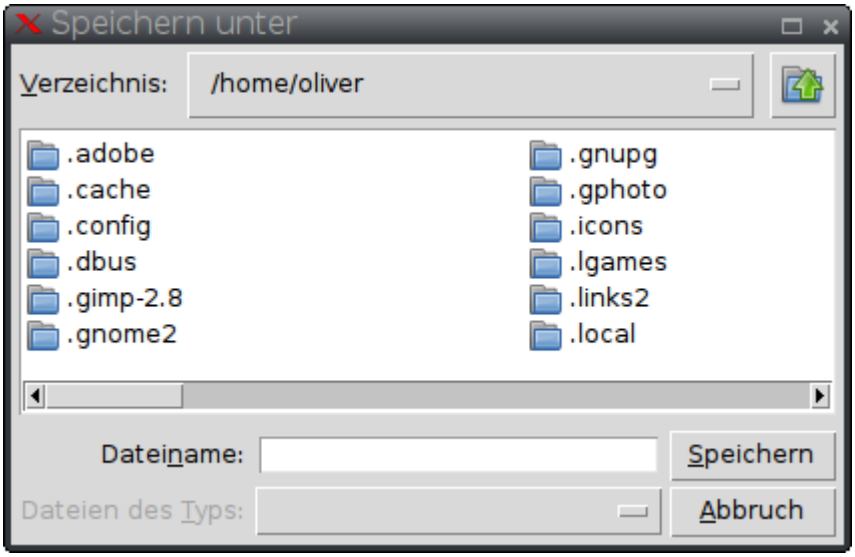
Der Befehl `tk_getSaveFile` zeigt einen Dialog zum Speichern einer Datei. Der Anwender soll einen Dateinamen eingeben. Wenn der Dateiname bereits existiert, fragt der Dialog, ob die Datei überschrieben werden soll. Der Rückgabewert ist ein Dateiname mit komplettem Pfad. Wenn der Dialog abgebrochen wird, wird ein leerer String zurückgegeben. Der Befehl ist fast identisch mit dem Befehl `tk_getOpenFile`.

Tabelle 40.4: Die wichtigsten Optionen

Option	Beschreibung
`-title "Titel"`	Titel
`-initialdir Ordner`	Ordner, der beim Öffnen des Dialogs bereits eingestellt ist
`-initialfile Dateiname`	Dateiname, der beim Öffnen des Dialogs als Dateiname vorgeschlagen wird
`-confirmoverwrite false`	Falls die Datei existiert, wird nicht nachgefragt, ob die Datei überschrieben werden soll
`-defaultextension Dateiendung`	Wenn der Anwender beim Dateinamen keine Dateiendung eingibt, wird diese Dateiendung automatisch hinzugefügt.

Listing 40.9: Eine Datei speichern (versteckte Ordner und Dateien werden angezeigt) (Beispiel212.tcl)

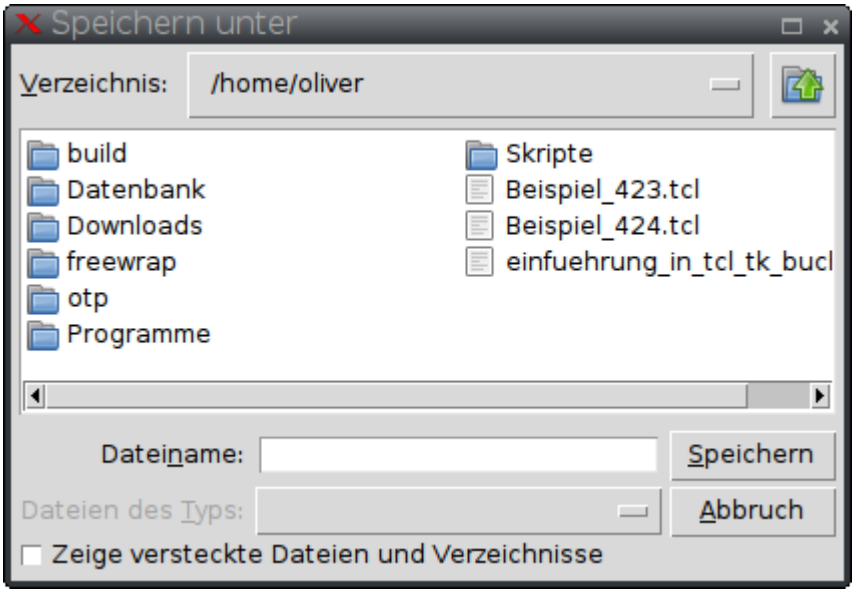
```
#!/usr/bin/env wish

set Dateiname [tk_getSaveFile]

if {$Dateiname != ""} {
  puts "$Dateiname wird gespeichert."
}
```



Das tatsächliche Speichern der Datei muss man noch programmieren (Zeile 6). Der Dialog gibt nur den Dateinamen zurück, unter dem die Datei gespeichert werden soll.

Listing 40.10: Eine Datei speichern (versteckte Ordner und Dateien sind ausgeblendet) (Beispiel424.tcl)

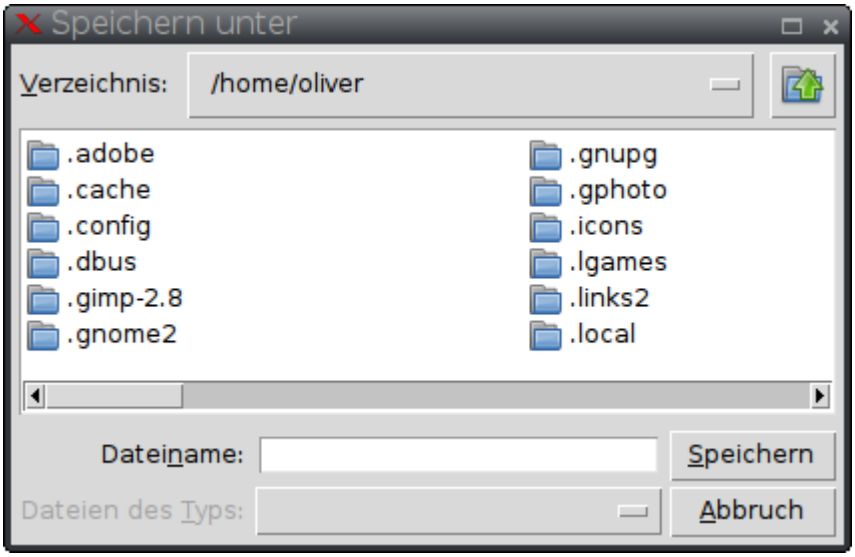
```
1  #!/usr/bin/env wish
2
3  catch {tk_getSaveFile foo bar}
4  set ::tk::dialog::file::showHiddenVar 0
5  set ::tk::dialog::file::showHiddenBtn 1
6
7  set Dateiname [tk_getSaveFile]
8
9  if {$Dateiname != ""} {
10   puts "$Dateiname wird gespeichert."
11 }
```



Der Dialog `tk_getSaveFile` bietet standardmäßig keine Option, um versteckte Ordner und Dateien auszublenden. Deshalb ist folgender Trick notwendig. In Zeile 3 wird der Dialog aufgerufen und sofort wieder geschlossen, ohne dass der Anwender den Dialog sieht. Dadurch werden alle mit dem Dialog im Hintergrund verbundenen Variablen initialisiert, so dass sie in den beiden folgenden Zeilen geändert werden können. In Zeile 4 wird festgelegt, dass die versteckten Ordner und Dateien ausgeblendet werden sollen. In Zeile 5 wird eingestellt, dass der Anwender eine Option angezeigt bekommt, um die versteckten Ordner und Dateien einzublenden.

Listing 40.11: Eine Datei speichern mit default-Dateiendung (Beispiel213.tcl)

```
1 #!/usr/bin/env wish
2
3 set Dateiname [tk_getSaveFile -defaultextension .txt]
4
5 if {$Dateiname != ""} {
6   puts "$Dateiname wird gespeichert."
7 }
```



Wenn der Anwender einen Dateinamen ohne Dateiendung eingibt (z. B. Brief statt Brief.txt), wird die default-Dateiendung automatisch ergänzt.

40.5 Ordner wählen-Dialog

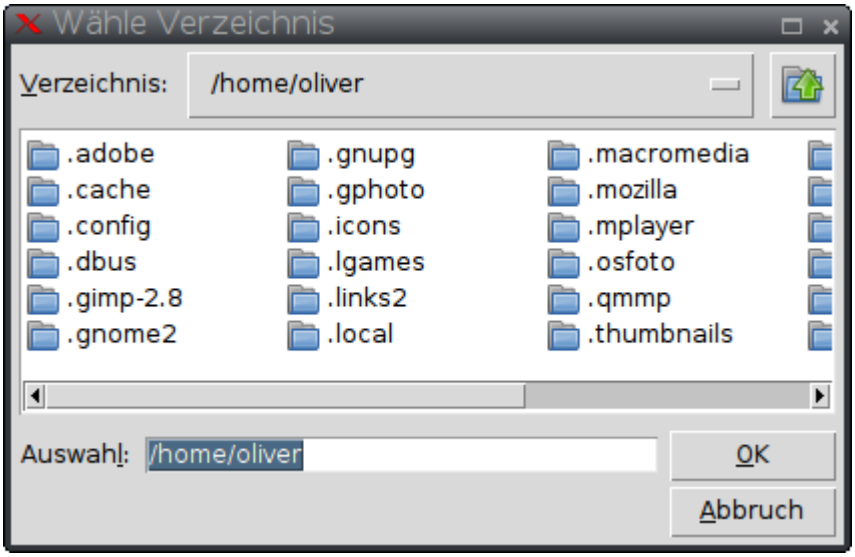
Der Befehl `tk_chooseDirectory` zeigt einen Dialog, um einen Ordner auszuwählen. Der Anwender darf auch einen (noch nicht) existierenden Ordner eingeben. Der Rückgabewert ist der Ordnername mit komplettem Pfad. Wenn der Dialog abgebrochen wird, wird ein leerer String zurückgegeben. Der Befehl ist ähnlich dem Befehl `tk_getOpenFile`.

Tabelle 40.5: Die wichtigsten Optionen

Option	Beschreibung
`-title "Titel"`	Titel
`-initialdir Ordner`	Ordner, der beim Öffnen des Dialogs bereits eingestellt ist
`-mustexist true`	Der Anwender kann nur Ordner auswählen, die bereits existieren.

Listing 40.12: Einen Ordner auswählen (Beispiel214.tcl)

```
#!/usr/bin/env wish

set Ordner [tk_chooseDirectory]

if {$Ordner != ""} {
  puts "Der Ordner ist $Ordner."
}
```



In Zeile 6 muss man noch programmieren, was mit dem Ordner gemacht werden soll.

41 Elemente

Die folgenden Unterkapitel stellen die einzelnen Elemente vor. Die Beispiele verwenden soweit möglich die neuen ttk-Elemente. Die Gestaltung der Elemente (d. h. der Stil der Elemente) wird in Kapitel 54 auf Seite 631 behandelt. In diesem Kapitel werden zunächst nur die Funktionsweisen der Elemente beschrieben.

41.1 Frame

Ein `frame`-Element ist ein Rahmen. Er dient dazu, mehrere Elemente zu gruppieren und damit die Anordnung der Elemente zu erleichtern. Das `frame`-Element kann mit und ohne Rahmen dargestellt werden. Der Rahmen wird automatisch genau so groß dargestellt, dass alle Element innerhalb des Rahmens hineinpassen. Außerdem kann man mit dem `frame`-Element auch eine Trennlinie erstellen.
Leider gibt es keine Möglichkeit, dem `frame`-Element Scroll-Leisten zuzuordnen. Wenn man bspw. viele Eingabefelder untereinander anordnet, die nicht mehr alle auf den Bildschirm passen, kann man das `frame`-Element nicht mit einer vertikalen Scroll-Leiste ausstatten. Man kann sich nur insofern helfen, dass man in das `frame`-Element ein `canvas`-Element setzt und dieses mit Scroll-Leisten verknüpft. In das `canvas`-Element werden dann die Eingabefelder platziert. Ein Beispiel hierzu finden Sie in Kapitel 51.11 auf Seite 617.

Tabelle 41.1: Die wichtigsten Optionen

Option	Beschreibung
`-borderwith Pixel`	Rahmenbreite

`-relief flat`
`-relief groove`
`-relief raised`
`-relief ridge`
`-relief solid`
`-relief sunken`
	3D-Effekt

`-padding {links oben rechts unten}`	Interner Abstand in Pixel (wenn man absolute Zahlen angibt)
`-padding [list $Links $Oben $Rechts $Unten]`	Interner Abstand in Pixel (wenn man Variablen verwendet)
`-width Pixel`	Breite (nur wenn man einen Rahmen als Trennlinie verwenden will)

Tabelle 41.1: Die wichtigsten Optionen

Option	Beschreibung
`-height Pixel`	Höhe (nur wenn man einen Rahmen als Trennlinie verwenden will)
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Rahmens
`-style Stil`	Stil

Da ein leerer Rahmen automatisch auf die Größe Null geschrumpft wird, platzieren die Beispiele ein `label`-Element in den Rahmen hinein. Dadurch hat der Rahmen die Größe des `label`-Elements.

Listing 41.1: Rahmen mit internem Abstand (absolute Zahlen) (Beispiel215.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::frame .fr -padding {30 20 10 0} -borderwidth 5 ↩
    -relief solid -cursor hand2
4 ttk::label .fr.lb -text "Text"
5 pack .fr.lb -side top
6 pack .fr
```

In Zeile 3 wurde bei der Option `-padding` der Abstand in absoluten Zahlen angegeben. Die Option erwartet eine Liste mit Werten. Wie man sieht hat das `label`-Element daraufhin einen Abstand zum Rand des Rahmens. Außerdem verändert sich durch die Option `-cursor` das Aussehen des Mauszeigers, wenn er über den Rahmen bewegt wird.

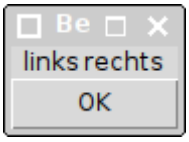
Listing 41.2: Rahmen mit internem Abstand (Variablen) (Beispiel216.tcl)

```
1 #!/usr/bin/env wish
2
3 set Links 30
4 set Oben 20
5 set Rechts 10
6 set Unten 0
7 ttk::frame .fr -padding [list $Links $Oben $Rechts $Unten]↩
     -borderwidth 5 -relief solid -cursor hand2
8 ttk::label .fr.lb -text "Text"
9 pack .fr.lb -side top
10 pack .fr
```

Wenn der interne Abstand des Rahmens über Variablen definiert werden soll, kann man bei der Option `-padding` keine geschweiften Klammern `{}` verwenden. Stattdessen wird der `list`-Befehl verwendet (Zeile 7).

Listing 41.3: Rahmen ohne Rand (Beispiel219.tcl)

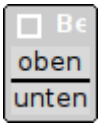
```
1  #!/usr/bin/env wish
2
3  ttk::frame .frRahmen
4  ttk::label .frRahmen.lbLabel1 -text "links"
5  ttk::label .frRahmen.lbLabel2 -text "rechts"
6  ttk::button .btButton -text "OK" -command {exit}
7
8  pack .frRahmen -side top
9  pack .frRahmen.lbLabel1 -side left
10 pack .frRahmen.lbLabel2 -side right
11 pack .btButton -side bottom
```



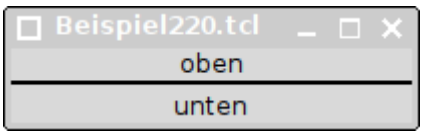
In obigem Beispiel dient der Rahmen dazu, zwei Labels aufzunehmen. Der Rahmen soll unsichtbar bleiben. Deshalb hat er keinen Rand.

Listing 41.4: Rahmen als waagrechte Trennlinie (Beispiel220.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::frame .frRahmen -width 1 -height 2 -borderwidth 1 ↩
     -relief solid
4  ttk::label .lbLabel1 -text "oben"
5  ttk::label .lbLabel2 -text "unten"
6
7  pack .lbLabel1 -side top
8  pack .frRahmen -side top -fill x
9  pack .lbLabel2 -side bottom
```



Wenn man das Fenster größer zieht:



In Zeile 3 wird ein Rahmen als Trennlinie definiert. Die Höhe (`-height`) muss mindestens 2 Pixel sein. Die Breite (`-width`) kann 1 Pixel sein, weil sich die Breite dynamisch mit der Fenstergröße ändert. Dies wird in Zeile 8 mit der Option `-fill x` festgelegt.

41.2 Label

Das `label`-Element zeigt einen Text oder ein Bild an.

Tabelle 41.2: Die wichtigsten Optionen

Option	Beschreibung
`-text "Text"`	Text
`-textvariable Variable`	Eine Variable, deren Inhalt im Label angezeigt wird
`-anchor nw`	Legt fest, wo der Text innerhalb des Elements platziert werden soll `n` = oben (north) `e` = rechts (east) `s` = unten (south) `w` = links (west) `center` = center (zentriert) Auch Kombinationen wie `nw` sind erlaubt.
`-wraplength Pixel`	Maximale Zeilenlänge in Pixel. Der Text wird dann in der nächsten Zeile fortgesetzt.

`-justify left`
`-justify center`
`-justify right`
	Ausrichtung des Textes, wenn er mehrzeilig ist
	`left`=linksbündig
	`center`=zentriert
	`right`=rechtsbündig

`-width AnzahlZeichen`	Breite

`-relief flat`
`-relief groove`
`-relief raised`
`-relief ridge`
`-relief solid`
`-relief sunken`
	3D-Effekt

Tabelle 41.2: Die wichtigsten Optionen

Option	Beschreibung
`-padding {links oben rechts unten}`	Interner Abstand in Pixel (wenn man absolute Zahlen angibt)
`-padding [list $Links $Oben $Rechts $Unten]`	Interner Abstand in Pixel (wenn man Variablen verwendet)
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil
`-image Bild`	Bild anzeigen. Unterstützte Dateiformate sind xbm, xpm, ppm, gif und (ab tcl 8.6) png

`-compound top`
`-compound bottom`
`-compound left`
`-compound right`
`-compound center`
	Position des Bildes bzw. Biltmaps in Bezug auf den Text
	`top`=oben
	`bottom`=unten
	`left`=links
	`right`=rechts
	`center`=mittig

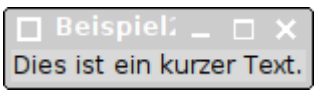
Die folgenden Optionen sollte man nur in Ausnahmefällen verwenden, weil man üblicherweise die Schrift und die Farben über den Stil festlegt.

Tabelle 41.3: Weitere Optionen

Option	Beschreibung
`-font Schrift`	Schrift
`-foreground Farbe`	Textfarbe
`-background Farbe`	Hintergrundfarbe

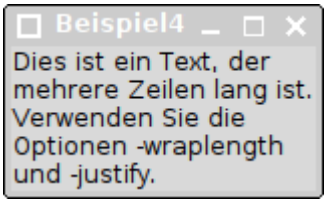
Listing 41.5: Label mit Text (Beispiel221.tcl)

```
#!/usr/bin/env wish

ttk::label .lb -text "Dies ist ein kurzer Text."
pack .lb
```



Listing 41.6: Label mit mehrzeiligem Text (Beispiel470.tcl)

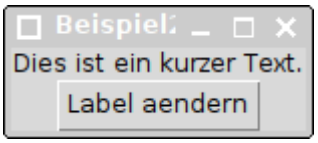
```
1 #!/usr/bin/env wish
2
3 ttk::label .lb -text "Dies ist ein Text, der mehrere Zeilen lang ist. Verwenden Sie die Optionen -wraplength und -justify." -wraplength 150 -justify left
4 pack .lb
```

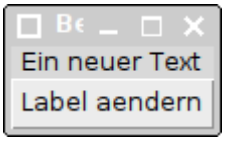


In Zeile 3 wurden die beiden Optionen `-wraplength` und `-justify` hinzugefügt. Die Option `-wraplength` bestimmt die Länge einer Textzeile (in Pixel) und die Option `-justify` legt die Ausrichtung des mehrzeiligen Textes (z. B. `left` = linksbündig) fest.

Listing 41.7: Label-Text ändern (mit Textvariable) (Beispiel223.tcl)

```
1 #!/usr/bin/env wish
2
3 proc TextAendern {} {
4   global Text
5   set Text "Ein neuer Text"
6   update idletasks
7 }
8
9 set Text "Dies ist ein kurzer Text."
10 ttk::label .lb -textvariable Text
11 ttk::button .bt -text "Label aendern" -command {TextAendern}
12 pack .lb
13 pack .bt
```

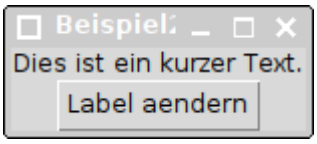


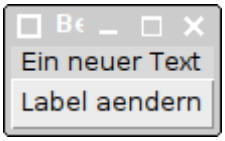


Durch die Verwendung einer Textvariablen kann man an beliebiger Stelle des Programms den im Label angezeigten Text verändern. Damit die Änderung des Textes sofort sichtbar wird, ist der Befehl `update idletasks` auszuführen.

Listing 41.8: Label-Text ändern (mit configure) (Beispiel224.tcl)

```
1  #!/usr/bin/env wish
2
3  proc TextAendern {} {
4    .lb configure -text "Ein neuer Text"
5    update idletasks
6  }
7
8  ttk::label .lb -text "Dies ist ein kurzer Text."
9  ttk::button .bt -text "Label aendern" -command TextAendern
10 pack .lb
11 pack .bt
```



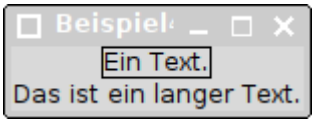


Auch mit dem Befehl `configure` in Zeile 4 kann man an einer beliebigen Stelle im Programm den Text (genauer gesagt: jede Eigenschaft eines Elements) ändern. Der Befehl wird in Kapitel 42.15 auf Seite 515 behandelt.

Die folgenden drei Beispiele zeigen die Anordnung des Textes mit Hilfe der Optionen `-anchor` und `-fill`. Dazu soll das obere `label`-Element rechtsbündig am unteren `label`-Element ausgerichtet werden.

Listing 41.9: Label-Text anordnen (Ausgangssituation) (Beispiel471.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::label .lb1 -text "Ein Text." -relief solid
4  ttk::label .lb2 -text "Das ist ein langer Text."
5  pack .lb1 -side top
6  pack .lb2 -side top
```

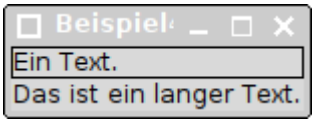


In der Ausgangssituation hat das obere `label`-Element mit der Option `-relief solid` einen Rahmen bekommen, um die Größe des Elements zu zeigen (Zeile 3).

Listing 41.10: Label-Text anordnen (Label waagrecht ausdehnen) (Beispiel472.tcl)

```
1  #!/usr/bin/env wish
```

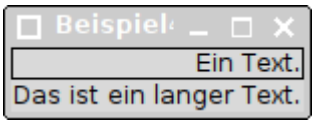
```
2
3 ttk::label .lb1 -text "Ein Text." -relief solid
4 ttk::label .lb2 -text "Das ist ein langer Text."
5 pack .lb1 -side top -fill x
6 pack .lb2 -side top
```



In Zeile 5 wurde durch die Option `-fill x` das `label`-Element waagrecht ausgedehnt, so dass es genauso breit wird, wie das untere `label`-Element.

Listing 41.11: Label-Text anordnen (Text rechtsbündig) (Beispiel473.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::label .lb1 -text "Ein Text." -relief solid -anchor e
4 ttk::label .lb2 -text "Das ist ein langer Text."
5 pack .lb1 -side top -fill x
6 pack .lb2 -side top
```



In Zeile 3 wird durch die Option `-anchor e` der Text rechtsbündig ausgerichtet. `e` steht für die Himmelsrichtung `east` (Osten).
Bei mehrzeiligem Text sind zusätzlich die Optionen `-wraplength` und `-justify` anzugeben.

41.3 Button

Ein Button ist ein Schaltknopf, der beim Anklicken einen Befehl ausführt. Im Folgenden werden nur die speziellen Eigenschaften des `Button`-Elements beschrieben. Die generellen Eigenschaften wurden bereits beim `Label`-Element vorgestellt.

Tabelle 41.4: Die wichtigsten Optionen

Option	Beschreibung
`-text "Text"`	Text
`-textvariable Variable`	Eine Variable, deren Inhalt im Label angezeigt wird
`-command Befehl`	Befehl, der ausgeführt wird, wenn der Anwender auf den Button klickt

Tabelle 41.4: Die wichtigsten Optionen

Option	Beschreibung
`-width AnzahlZeichen`	Breite
`-state normal` `-state disabled`	Status des Elements. `normal` = Element kann angeklickt werden `disabled` = Element kann nicht angeklickt werden

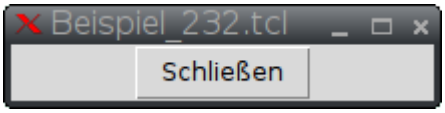
`-default normal`
`-default active`
`-default disabled`
	Legt fest, ob der Button ausgeführt wird, wenn der Benutzer die Return-Taste betätigt.

`-padding {links oben rechts unten}`	Interner Abstand in Pixel (wenn man absolute Zahlen angibt)
`-padding [list $Links $Oben $Rechts $Unten]`	Interner Abstand in Pixel (wenn man Variablen verwendet)
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil
`-image Bild`	Bild anzeigen. Unterstützte Dateiformate sind xbm, xpm, ppm, gif und (ab tcl 8.6) png

`-compound top`
`-compound bottom`
`-compound left`
`-compound right`
`-compound center`
	Position des Bildes bzw. Biltmaps in Bezug auf den Text
	`top`=oben
	`bottom`=unten
	`left`=links
	`right`=rechts
	`center`=mittig

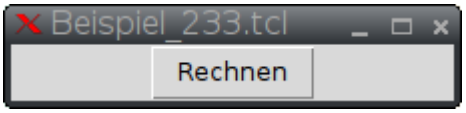
Listing 41.12: Button mit exit-Befehl (Beispiel232.tcl)

```
#!/usr/bin/env wish

ttk::button .bt -text "Schliessen" -command {exit}
pack .bt
```

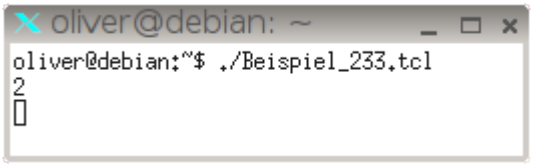


Listing 41.13: Button, der eine Prozedur aufruft (Beispiel233.tcl)

```
#!/usr/bin/env wish

proc Rechnen {} {
  puts [expr 1 + 1]
}

ttk::button .bt -text "Rechnen" -command {Rechnen}
pack .bt
```



Wenn der Button angeklickt wurde:



Listing 41.14: Mehrere gleichartige Buttons mit for-Schleife erzeugen (so funktioniert es nicht) (Beispiel388.tcl)

```
1 #!/usr/bin/env wish
2
3 for {set i 1} {$i <= 3} {incr i} {
4   ttk::button .btButton$i -text "Button $i" -command {set ↩
      LabelText "Button $i";update idletasks}
5   pack .btButton$i -side top
6 }
7 set i 99
8 ttk::label .lbLabel -textvariable LabelText
9 pack .lbLabel -side top
```

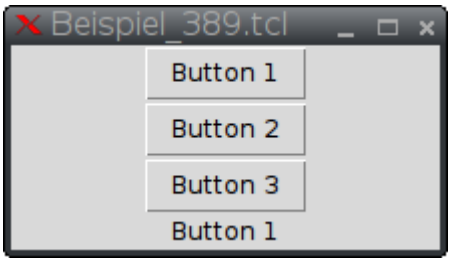
In den Zeilen 3 bis 6 werden drei Buttons erzeugt. Über die Variable `$i` bekommen die Buttons die Name `.btButton1`, `.btButton2` und `.btButton3`. Die Beschriftung der Buttons ist entsprechend `Button 1`, `Button 2` und `Button 3`. Etwas schwieriger ist es, den Button drei unterschiedliche Befehle zuzuordnen. Da der Befehl in geschweifte

Klammern `{}` gesetzt wurde, wird die Variable `$i` nicht interpretiert während der Button erzeugt wird. Jeder Button bekommt deshalb folgenden Befehl zugeordnet:
`set LabelText "Button $i";update idletasks`
Erst wenn der Button zur Programmlaufzeit vom Anwender angeklickt wird, wird die Variable `$i` durch den dann gültigen Wert für `i` ersetzt. Da in der Zeile 7 die Variable `i` auf `99` gesetzt wird, erscheint beim Klick auf einen Button der Text `Button 99`.

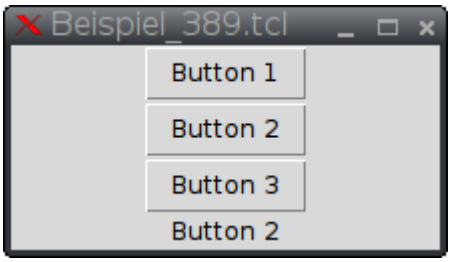
Listing 41.15: Mehrere gleichartige Buttons mit for-Schleife erzeugen (jetzt funktioniert es) (Beispiel389.tcl)

```
1 #!/usr/bin/env wish
2
3 for {set i 1} {$i <= 3} {incr i} {
4   ttk::button .btButton$i -text "Button $i" -command "set ↲
      LabelText \"Button $i\";update idletasks"
5   pack .btButton$i -side top
6 }
7 set i 99
8 ttk::label .lbLabel -textvariable LabelText
9 pack .lbLabel -side top
```

Wenn der Button 1 angeklickt wird:



Wenn der Button 2 angeklickt wird:



In Zeile 4 wird der Befehl für die Buttons in Anführungszeichen `""` statt in geschweifte Klammern `{}` gesetzt. Dadurch wird die Variable `$i` bereits ersetzt, wenn die Buttons erzeugt werden, und die Befehle enthalten somit keine Variable mehr (siehe auch Kapitel 10.2). Die Befehle lauten:
`set LabelText "Button 1";update idletasks` für den Button 1
`set LabelText "Button 2";update idletasks` für den Button 2
`set LabelText "Button 3";update idletasks` für den Button 3
Da der Befehl in Anführungszeichen gesetzt wurde, muss man die Anführungszeichen innerhalb des Befehls mit einem Backslash \ maskieren.

41.4 Entry

Ein `entry`-Element ist ein Eingabefeld für Text oder Zahlen. Der Inhalt des Felds wird in einer Variablen gespeichert. Diese Variable ist immer global, so dass man Acht geben muss, keinen Namenskonflikt im Programm zu bekommen.

Tabelle 41.5: Die wichtigsten Optionen

Option	Beschreibung
`-textvariable Variable`	Eine Variable, die den Inhalt des Entry-Elements speichert.
`-width AnzahlZeichen`	Breite

`-state normal`
`-state disabled`
`-state readonly`
	Status des Elements.
	`normal` = Eingabe ist möglich
	`disabled` = keine Eingabe möglich. Der Text kann nicht selektiert werden.
	`readonly` = wie `disabled`, aber der Text kann selektiert werden

`-show *`	Zeigt anstatt der Eingabe nur Sternchen. Dies ist z. B. bei der Passworteingabe nützlich.
`-xscrollcommand`	Das Feld wird mit einer horizontalen Scroll-Leiste verknüpft
`-validatecommand Prozedur`	Legt fest, welche Prozedur den Eingabewert überprüft. Die Prozedur muss einen Boolean-Wert zurückgeben (`0`=false, `1`=true). Die Prozedur kann z.B. prüfen, dass nur Ziffern eingegeben werden.

`-validate none`
`-validate focus`
`-validate focusin`
`-validate focusout`
`-validate key`
`-validate all`
	Legt fest, bei welchem Ereignis der Eingabewert überprüft werden soll (also validatecommand aufgerufen werden soll). Die Beschreibung der Ereignisse erfolgt in nachstehender Tabelle.

`-justify left`
`-justify center`
`-justify right`
	Ausrichtung des Textes
	`left`=linksbündig
	`center`=zentriert
	`right`=rechtsbündig

`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements

Tabelle 41.5: Die wichtigsten Optionen

Option	Beschreibung
`-style Stil`	Stil

Tabelle 41.6: Befehle

Befehl	Beschreibung
`Elementname insert Index Text`	Fügt Text ein. Der Index gibt die Stelle an. Das erste Zeichen hat den Index 0, das letzte Zeichen hat den Index `end`.
`Elementname delete IndexVon IndexBis`	Löscht Text.
`Elementname icursor Index`	Platziert den Eingabecursor an die Stelle vor Index.
`Elementname selection range IndexVon IndexBis`	Markiert den Text.

Tabelle 41.7: Validate-Ereignisse

Ereignis	Beschreibung
`-validate none`	keine überprüfung
`-validate focus`	Das Eingabefeld erhält oder verliert den Fokus.
`-validate focusin`	Das Eingabefeld erhält den Fokus.
`-validate focusout`	Das Eingabefeld verliert den Fokus.
`-validate key`	Es erfolgt eine Eingabe.
`-validate all`	Alle Ergeignisse

Tabelle 41.8: Die wichtigsten Platzhalter während der Validierung

Platzhalter	Beschreibung
`%P`	die gesamte Eingabe im Eingabefeld
`%S`	das zuletzt eingegebene Zeichen

Tabelle 41.8: Die wichtigsten Platzhalter während der Validierung

Platzhalter	Beschreibung
`%i`	der Index des Zeichens

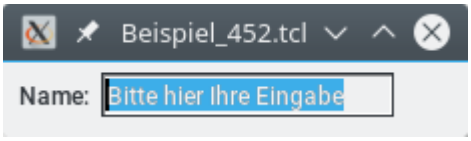
Listing 41.16: Eingabefeld (Beispiel234.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::entry .enText -textvariable Text
4 pack .enText
5 focus .enText
```

In Zeile 5 wird der Fokus auf das Eingabefeld gesetzt.

Listing 41.17: Entry mit Text vorbelegen und markieren (Beispiel452.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::label .lb -text "Name:"
4 ttk::entry .en
5 pack .lb -side left -padx 2
6 pack .en -side left -padx 2
7
8 .en insert end "Bitte hier Ihre Eingabe"
9 focus .en
10 .en selection range 0 end
```



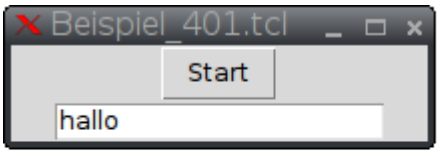
In Zeile 4 wird ein `Entry`-Element erzeugt. In Zeile 8 wird ein Text eingefügt. In Zeile 9 bekommt das Element den Fokus. In Zeile 10 wird der Text selektiert, so dass der Anwender durch seine Tastatureingabe den vorbelegten Text überschreibt.

Listing 41.18: Die Textvariable des Entry-Elements ist immer global (Beispiel401.tcl)

```
1 #!/usr/bin/env wish
2
3 proc Modaldialog {} {
4   toplevel .mod
5   ttk::label .mod.lb -text "Eingabe:"
6   ttk::entry .mod.en -textvariable Text
```

```
7   ttk::button .mod.bt -text "OK" -command {grab release ↩
       .mod; destroy .mod}
8   pack .mod.lb -side left
9   pack .mod.en -side left
10  pack .mod.bt -side left
11  focus .mod.en
12  tkwait visibility .mod
13  grab set .mod
14 }
15
16 set Text ""
17 ttk::button .bt -text "Start" -command {Modaldialog}
18 ttk::entry .en -textvariable Text
19 pack .bt
20 pack .en
```

Wenn man im ersten Fenster auf den Button Start klickt und dann im zweiten Fenster einen Text eingibt, erscheint der Text unmittelbar auch im ersten Fenster.



Sowohl im Hauptprogramm als auch in der Prozedur werden Entry-Elemente platziert. Beide Element bekommen dieselbe Textvariable `Text` zugewiesen. Da die Textvariable des Entry-Elements immer global ist (auch ohne dass der Befehl `global` in der Prozedur verwendet wird), zeigen beide Elemente jederzeit den gleichen Inhalt an.

Listing 41.19: Eingabefeld und Button (ohne Prozedur) (Beispiel387.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::entry .enText -textvariable EntryText
4  ttk::label .lbLabel -textvariable LabelText
5  ttk::button .btButton -text "OK" -command {set LabelText ↩
      $EntryText; update idletasks}
6
7  pack .enText -side top
8  pack .lbLabel -side top
9  pack .btButton -side bottom
10 focus .enText
```

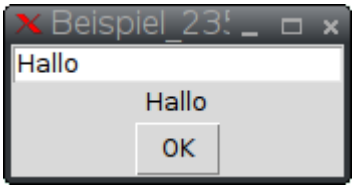
Ergebnis nach Eingabe `Hallo` und Klick auf den Button:

In Zeile 5 wird mit dem Parameter `-command` festgelegt, welche Befehle beim Klick auf den Button ausgeführt werden sollen. Der Befehl `update idletasks` ist notwendig, damit die Bildschirmausgabe aktualisiert wird.

Listing 41.20: Eingabefeld und Button (mit Prozedur, komplizierte Lösung) (Beispiel235.tcl)

```
1  #!/usr/bin/env wish
2
3  proc LabelTextSetzen {tmp1 tmp2} {
4    upvar $tmp1 Label
5    upvar $tmp2 Entry
6
7    set Label $Entry
8    update idletasks
9  }
10
11 ttk::entry .enText -textvariable EntryText
12 ttk::label .lbLabel -textvariable LabelText
13 button .btButton -text "OK" -command {LabelTextSetzen ↩
     LabelText EntryText }
14
15 pack .enText -side top
16 pack .lbLabel -side top
17 pack .btButton -side bottom
18 focus .enText
```

Ergebnis nach Eingabe `Hallo` und Klick auf den Button:



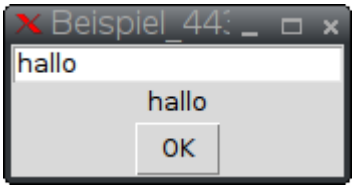
In diesem Beispiel wird der Befehl, der beim Klicken auf den Button ausgeführt werden soll, in eine separate Prozedur ausgelagert. Da der Inhalt aus dem Eingabefeld in das `Label`-Element übernommen werden soll, müssen die Textvariablen der beiden Elemente an die Prozedur übergeben werden. In Zeile 13 werden an die Prozedur die Namen der beiden Textvariablen (vom Eingabefeld und vom Label) übergeben. Innerhalb der Prozedur wird dann mit `upvar` ein Verweis auf diese beiden Variablen erzeugt (Zeilen 4 und 5).

Listing 41.21: Eingabefeld und Button (mit Prozedur, einfache Lösung) (Beispiel443.tcl)

```
1  #!/usr/bin/env wish
2
```

```
3  proc LabelTextSetzen {Text} {
4    .lbLabel configure -text $Text
5    update idletasks
6  }
7
8  ttk::entry .enText -textvariable EntryText
9  ttk::label .lbLabel -text ""
10 ttk::button .btButton -text "OK" -command {LabelTextSetzen
     $EntryText}
11
12 pack .enText -side top
13 pack .lbLabel -side top
14 pack .btButton -side bottom
15 focus .enText
```

Ergebnis nach Eingabe `Hallo` und Klick auf den Button:



Anstatt wie im Beispiel zuvor die Namen der Textvariablen an die Prozedur zu übergeben und mit dem Befehl `upvar` auf den Inhalt dieser Variablen zuzugreifen, wird das Label jetzt ohne Textvariable definiert (Zeile 9). In Zeile 10 wird im `command`-Befehl die Prozedur aufgerufen und der Inhalt der Variable `EntryText` an die Prozedur übergeben. In Zeile 4 wird dann der Text des Labels mit dem Befehl `configure` geändert (der Befehl wird in Kapitel 42.15 auf Seite 515 beschrieben).

Listing 41.22: Überprüfung der Eingabe mit einer Prozedur (Beispiel236.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Pruefen {Text Zeichen Index} {
4    puts "$Text | $Zeichen | $Index"
5    return 1
6  }
7
8  ttk::entry .enText -textvariable Text -validate key
     -validatecommand {Pruefen %P %S %i}
9  pack .enText
10 focus .enText
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_236.tcl
a | a | 0
ab | b | 1
abc | c | 2
abcd | d | 3
abcde | e | 4
```

In Zeile 8 legt die Option `-validate key` fest, dass sofort jedes Zeichen überprüft wird. Die Option `-validatecommand` ruft die Validierungsprozedur auf. In diesem Beispiel wird allerdings innerhalb der Prozedur nichts geprüft, sondern lediglich eine Ausgabe gemacht, um zu zeigen, wie man den Inhalt des Eingabefelds entgegen nimmt.

Listing 41.23: Beispielhafte Überprüfung der Eingabe (Beispiel237.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Pruefen {Text Zeichen Index} {
4    #puts "$Text | $Zeichen | $Index"
5
6    # nur Ziffern
7    return [string is integer $Zeichen]
8
9    # nur Buchstaben
10   #return [string is alpha $Zeichen]
11
12   # maximal 5 Zeichen
13   #if {$Index <= 4} {return 1} {return 0}
14
15   # maximal 5 Buchstaben
16   #if {$Index <= 4 && [string is alpha $Zeichen] == 1} {
       return 1} {return 0}
17
18   # nur Zahlen 0 bis 99
19   #if {$Index <= 1 && [string is integer $Zeichen] == 1} {
       return 1} {return 0}
20 }
21
22 ttk::entry .enText -textvariable Text -validate key
     -validatecommand {Pruefen %P %S %i}
23 pack .enText
24 focus .enText
```

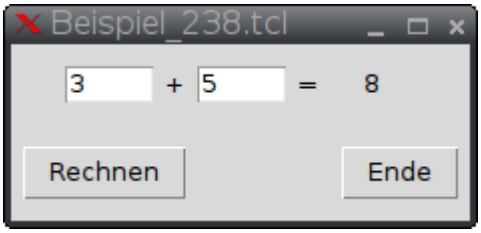
In der Prozedur `Pruefen` sind beispielhaft verschiedene Prüfroutinen dargestellt.

41.5 Frame, Label, Entry und Button zusammen

Ein Beispiel, wie man Frame, Label, Entry und Button zu einem kleinen Rechenprogramm zusammenfügt.

Listing 41.24: Frame, Label, Entry und Button zusammen (Beispiel238.tcl)

```
#!/usr/bin/env wish

proc Pruefen {Text Zeichen Index} {
  return [string is integer $Zeichen]
}

proc Rechnen {Zahl1 Zahl2} {
  if {$Zahl1 != "" && $Zahl2 != ""} {
    set Ergebnis [expr $Zahl1 + $Zahl2]
  } else {
    set Ergebnis "Fehler"
  }
  return $Ergebnis
}

ttk::frame .frAufgabe
ttk::frame .frButton

ttk::entry .frAufgabe.enZahl1 -textvariable Zahl1 -width 5 -validate key -validatecommand {Pruefen %P %S %i}
ttk::entry .frAufgabe.enZahl2 -textvariable Zahl2 -width 5 -validate key -validatecommand {Pruefen %P %S %i}
ttk::label .frAufgabe.lbErgebnis -textvariable Ergebnis -width 5

ttk::label .frAufgabe.lbPlus -text "+"
ttk::label .frAufgabe.lbGleich -text "="

ttk::button .frButton.btRechnen -text "Rechnen" -command {set Ergebnis [Rechnen $Zahl1 $Zahl2]}
ttk::button .frButton.btEnde -text "Ende" -command exit

pack .frAufgabe -side top -padx 5 -pady 10
pack .frAufgabe.enZahl1 -side left
pack .frAufgabe.lbPlus -side left -padx 3
pack .frAufgabe.enZahl2 -side left
pack .frAufgabe.lbGleich -side left -padx 3
pack .frAufgabe.lbErgebnis -side left

pack .frButton -side bottom -fill x -pady 10
pack .frButton.btRechnen -side left -padx 5
pack .frButton.btEnde -side right -padx 5
focus .frAufgabe.enZahl1
```



41.6 Labelframe

Das Labelframe-Element ist ein Rahmen, der zusätzlich eine Bezeichnung (Label) hat. Die Bezeichnung kann man entweder über die Option `-text` oder `-labelwidget` zuordnen. Es ist ratsam, die Option `-labelwidget` zu verwende, weil dann die Schriftgröße automatisch angepasst wird, wenn sich diese generell im Programm ändert.

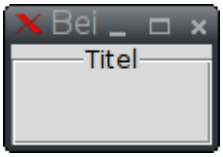
Tabelle 41.9: Die wichtigsten Optionen

Option	Beschreibung
`-text "Text"`	Beschriftung des Rahmens (es ist aber besser, die Option `-labelwidget` zu verwenden)
`-labelwidget`	Beschriftung des Rahmens mit einem Label-Element
`-labelanchor n/ne/e/se/s/sw/w/nw`	Position der Rahmenbeschriftung gemäß der Himmelsrichtungen Norden (n), Osten (e=east), Süden (s) oder Westen (w).
`-width AnzahlZeichen`	Breite
`-height Pixel`	Höhe
`-padding {links oben rechts unten}`	Interner Abstand in Pixel (wenn man absolute Zahlen angibt)
`-padding [list $Links $Oben $Rechts $Unten]`	Interner Abstand in Pixel (wenn man Variablen verwendet)
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil

Listing 41.25: Labelframe mit der Option -text (Beispiel239.tcl)

```
#!/usr/bin/env wish

ttk::labelframe .lbfLabel -text Titel -labelanchor n ↩
    -width 100 -height 50
```

```
4 pack .lbfLabel
```



Listing 41.26: Labelframe mit der Option -labelwidget (Beispiel428.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::label .lb -text "Titel"
4 ttk::labelframe .lbframe -labelwidget .lb
5 ttk::label .lbframe.lb -text "Ein Label"
6
7 pack .lbframe -pady 3
8 pack .lbframe.lb -pady 3
9 pack .lbframe -pady 3
```

In Zeile 3 wird ein Label definiert. In Zeile 4 wird das Label dem Labelframe zugewiesen.

41.7 Checkbutton

Ein Checkbutton ist ein Ankreuzfeld. Es kennt zwei Zustände: markiert und nicht markiert.

Tabelle 41.10: Die wichtigsten Optionen

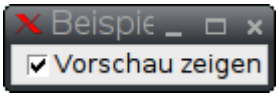
Option	Beschreibung
`-text "Text"`	Beschriftung des Checkbuttons
`-textvariable Variable`	Eine Variable, dessen Inhalt als Beschriftung des Checkbuttons dient.
`-variable Variable`	Variable, die den Zustand des Checkbuttons speichert. `0` = nicht markiert, `1` = markiert.
`-command`	Befehl, der ausgeführt wird, wenn der Checkbutton angeklickt wird
`-width AnzahlZeichen`	Breite

Tabelle 41.10: Die wichtigsten Optionen

Option	Beschreibung
`-state normal` `-state disabled`	Status des Elements. `normal` = Element kann angeklickt werden `disabled` = Element kann nicht angeklickt werden
`-padding {links oben rechts unten}`	Interner Abstand in Pixel (wenn man absolute Zahlen angibt)
`-padding [list $Links $Oben $Rechts $Unten]`	Interner Abstand in Pixel (wenn man Variablen verwendet)
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil
`-image Bild`	Bild
`-compound top/bottom/left/right/center`	Position des Bildes bzw. Bitmaps in Bezug auf den Text (oben / unten / links / rechts / mittig)

Listing 41.27: Checkbutton (Beispiel240.tcl)

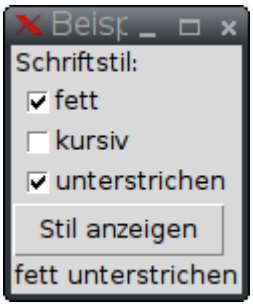
```
#!/usr/bin/env wish

ttk::checkbutton .cbVorschau -text "Vorschau zeigen" ↩
    -variable Vorschau
pack .cbVorschau
```



Listing 41.28: Checkbuttons auswerten (Beispiel241.tcl)

```
#!/usr/bin/env wish

proc StilAnzeigen {Fett Kursiv Unterstrichen} {
  set Auswahl ""
  if {$Fett == 1} {set Auswahl "$Auswahl fett"}
  if {$Kursiv == 1} {set Auswahl "$Auswahl kursiv"}
  if {$Unterstrichen == 1} {set Auswahl "$Auswahl ↩
     unterstrichen"}
  return $Auswahl
}
```

```
10
11 set Auswahl ""
12 set Fett 0
13 set Kursiv 1
14 set Unterstrichen 0
15 ttk::label .lbStil -text "Schriftstil:"
16 ttk::checkbutton .cbFett -text "fett" -variable Fett
17 ttk::checkbutton .cbKursiv -text "kursiv" -variable Kursiv
18 ttk::checkbutton .cbUnterstrichen -text "unterstrichen" ↩
     -variable Unterstrichen
19 ttk::button .btStilAnzeigen -text "Stil anzeigen" -command↩
      {set Auswahl [StilAnzeigen $Fett $Kursiv ↩
     $Unterstrichen]}
20 ttk::label .lbAuswahl -textvariable Auswahl
21
22 pack .lbStil -side top -anchor w
23 pack .cbFett -side top -anchor w
24 pack .cbKursiv -side top -anchor w
25 pack .cbUnterstrichen -side top -anchor w
26 pack .btStilAnzeigen -side top -anchor w
27 pack .lbAuswahl -side top -anchor w
```

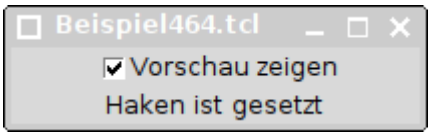


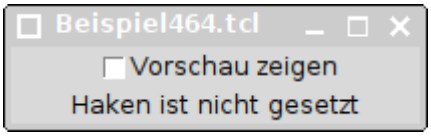
In den Zeilen 12 bis 14 werden die anfänglichen Zustande der Checkbuttons festgelegt. Der Checkbutton `kursiv` wird beim Programmstart markiert.

Listing 41.29: Checkbutton führt einen Befehl aus (Beispiel464.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Klick {Wert} {
4          global Text
5          if {$Wert == 1} {
6                  set Text "Haken ist gesetzt"
7          } else {
8                  set Text "Haken ist nicht gesetzt"
9          }
10         update idletasks
11 }
12
13 set Text ""
14 ttk::checkbutton .cbVorschau -text "Vorschau zeigen" ↩
     -variable Vorschau -command {Klick $Vorschau}
```

```
15 ttk::label .lbAnzeige -textvariable Text
16 pack .cbVorschau
17 pack .lbAnzeige
```





In Zeile 14 wird durch die Option `-command` ein Befehl ausgeführt, sobald man mit der Maus auf den Checkbutton klickt.

41.8 Radiobutton

Radiobuttons sind alternativ auswählbare Elemente, d. h. es kann immer nur ein Radiobutton ausgewählt werden. Die anderen Radiobuttons sind automatisch nicht ausgewählt.

Tabelle 41.11: Die wichtigsten Optionen

Option	Beschreibung
`-text "Text"`	Beschriftung des Radiobuttons
`-textvariable Variable`	Eine Variable, dessen Inhalt als Beschriftung des Radiobuttons dient.
`-value Wert`	Wert des Radiobuttons
`-variable Variable`	Variable, die den Zustand des Radiobuttons speichert. `0` = nicht markiert `1` = markiert.
`-command`	Befehl, der ausgeführt wird, wenn der Radiobutton angeklickt wird
`-width AnzahlZeichen`	Breite
`-state normal` `-state disabled`	Status des Elements. `normal` = Element kann angeklickt werden `disabled` = Element kann nicht angeklickt werden

Tabelle 41.11: Die wichtigsten Optionen

Option	Beschreibung
`-padding {links oben rechts unten}`	Interner Abstand in Pixel (wenn man absolute Zahlen angibt)
`-padding [list $Links $Oben $Rechts $Unten]`	Interner Abstand in Pixel (wenn man Variablen verwendet)
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil
`-image Bild`	Bild
`-compound top/bottom/left/right/center`	Position des Bildes bzw. Bitmaps in Bezug auf den Text (oben / unten / links / rechts / mittig)

Listing 41.30: Radiobuttons (Beispiel242.tcl)

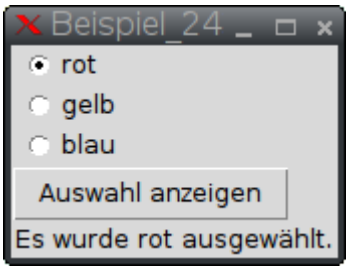
```
1  #!/usr/bin/env wish
2
3  set Farbe "rot"
4  ttk::radiobutton .rbRot -text "rot" -variable Farbe -value "rot"
5  ttk::radiobutton .rbGelb -text "gelb" -variable Farbe -value "gelb"
6  ttk::radiobutton .rbBlau -text "blau" -variable Farbe -value "blau"
7
8  pack .rbRot -side top -anchor w
9  pack .rbGelb -side top -anchor w
10 pack .rbBlau -side top -anchor w
```

In den Zeilen 4 bis 6 werden die Radiobuttons definiert. Wichtig ist, dass man allen drei Radiobuttons dieselbe Variable zuordnet.

Listing 41.31: Wert des ausgewählten Radiobuttons anzeigen (Beispiel243.tcl)

```
1  #!/usr/bin/env wish
2
3  proc AuswahlAnzeigen {Farbe} {
4    if {$Farbe != ""} {
```

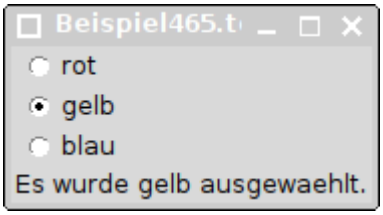
```
5       set Auswahl "Es wurde $Farbe ausgewaehlt."
6     } else {
7       set Auswahl "Es wurde nichts ausgewaehlt."
8     }
9     return $Auswahl
10 }
11
12 set Farbe "rot"
13 ttk::radiobutton .rbRot -text "rot" -variable Farbe -value
       "rot"
14 ttk::radiobutton .rbGelb -text "gelb" -variable Farbe
      -value "gelb"
15 ttk::radiobutton .rbBlau -text "blau" -variable Farbe
      -value "blau"
16 ttk::button .btZeigen -text "Auswahl anzeigen" -command {
      set Auswahl [AuswahlAnzeigen $Farbe]}
17 ttk::label .lbAuswahl -textvariable Auswahl
18
19 pack .rbRot -side top -anchor w
20 pack .rbGelb -side top -anchor w
21 pack .rbBlau -side top -anchor w
22 pack .btZeigen -side top -anchor w
23 pack .lbAuswahl -side top -anchor w
```



Listing 41.32: Radiobutton führt einen Befehl aus (Beispiel465.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Klick {Wert} {
4          global Text
5          set Text "Es wurde $Wert ausgewaehlt."
6          update idletasks
7  }
8
9  set Text ""
10 set Farbe "rot"
11 ttk::radiobutton .rbRot -text "rot" -variable Farbe -value
       "rot" -command {Klick $Farbe}
12 ttk::radiobutton .rbGelb -text "gelb" -variable Farbe
      -value "gelb" -command {Klick $Farbe}
13 ttk::radiobutton .rbBlau -text "blau" -variable Farbe
      -value "blau" -command {Klick $Farbe}
14 ttk::label .lbAnzeige -textvariable Text
```

```
15
16 pack .rbRot -side top -anchor w
17 pack .rbGelb -side top -anchor w
18 pack .rbBlau -side top -anchor w
19 pack .lbAnzeige
```



In den Zeilen 11 bis 13 wird durch die Option `-command` ein Befehl ausgeführt, sobald man mit der Maus auf den Radiobutton klickt.

41.9 Menubutton

Ein Menubutton ist ein Button, der ein Menü anzeigt, wenn man ihn anklickt. Wie man ein Menü erstellt, wird in Kapitel 43 auf Seite 521 ausführlich behandelt. Der Vollständigkeit halber soll der Menubutton bereits an dieser Stelle vorgestellt werden.

Tabelle 41.12: Die wichtigsten Optionen

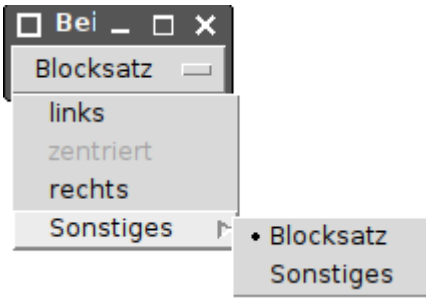
Option	Beschreibung
`-text "Text"`	Beschriftung des Menubuttons
`-textvariable Variable`	Eine Variable, dessen Inhalt als Beschriftung des Menubuttons dient.
`-direction above/below/ left/right/flush`	Position des Menüs (oben / unten / links / rechts / mittig)
`-width AnzahlZeichen`	Breite
`-state normal` `-state disabled`	Status des Elements. `normal` = Element kann angeklickt werden `disabled` = Element kann nicht angeklickt werden
`-padding {links oben rechts unten}`	Interner Abstand in Pixel (wenn man absolute Zahlen angibt)
`-padding [list $Links $Oben $Rechts $Unten]`	Interner Abstand in Pixel (wenn man Variablen verwendet)
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements

Tabelle 41.12: Die wichtigsten Optionen

Option	Beschreibung
`-style Stil`	Stil

Listing 41.33: Menubutton (Beispiel475.tcl)

```
1  #!/usr/bin/env wish
2
3  set Ausrichtung "links"
4  ttk::menubutton .mb -textvariable Ausrichtung
5  menu .mb.menu -tearoff 0
6  .mb configure -menu .mb.menu
7  .mb.menu add radiobutton -value "links" -variable ↩
     Ausrichtung -label "links"
8  .mb.menu add radiobutton -value "zentriert" -variable ↩
     Ausrichtung -label "zentriert" -state disabled
9  .mb.menu add radiobutton -value "rechts" -variable ↩
     Ausrichtung -label "rechts"
10 .mb.menu add cascade -label "Sonstiges" -menu ↩
     .mb.menu.sonstiges
11 menu .mb.menu.sonstiges -tearoff 0
12 .mb.menu.sonstiges add radiobutton -value "Blocksatz" ↩
     -variable Ausrichtung -label "Blocksatz"
13 .mb.menu.sonstiges add radiobutton -value "Sonstiges" ↩
     -variable Ausrichtung -label "Sonstiges"
14
15 pack .mb -side left
```



Die Zeilen 5 bis 13 erstellen ein Menü, das zu dem Menubutton gehört (siehe Kapitel 43 auf Seite 521).

41.10 Spinbox

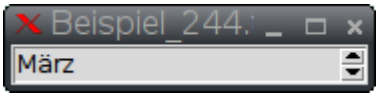
Eine Spinbox dient der Auswahl aus einer Liste von vorgegebenen Werten.

Tabelle 41.13: Die wichtigsten Optionen

Option	Beschreibung
`-values Liste`	Liste der Auswahlmöglichkeiten
`-textvariable Variable`	das ausgewählte Listenelement
`-state readonly` `-state normal`	`readonly` = Der Anwender darf nur Listenelemente auswählen und keine beliebige Eingabe machen `normal` = Der Anwender kann ein Listenelement auswählen oder eine beliebige Eingabe machen.
`-from`	Startwert
`-to`	Endwert
`-increment`	Erhöhung des Wertes
`-command`	Befehl, der ausgeführt wird, wenn die Spinbox angeklickt wird
`-width AnzahlZeichen`	Breite
`-state normal` `-state disabled`	Status des Elements. `normal` = Element kann angeklickt werden `disabled` = Element kann nicht angeklickt werden
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil

Listing 41.34: Spinbox (Beispiel244.tcl)

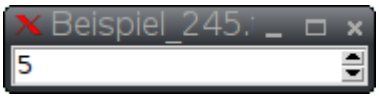
```
#!/usr/bin/env wish

set Monat Januar
set Liste {Januar Februar Maerz April Mai Juni}
ttk::spinbox .sbMonat -values $Liste -textvariable Monat ↩
   -state readonly
pack .sbMonat
```

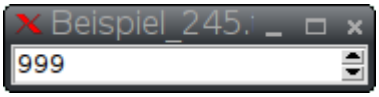


Durch die Option `-state readonly` kann der Anwender nur einen Eintrag aus den Listenwerten auswählen.

Listing 41.35: Auswahl aus einem Zahlenbereich kombiniert mit beliebiger Eingabe (Beispiel245.tcl)

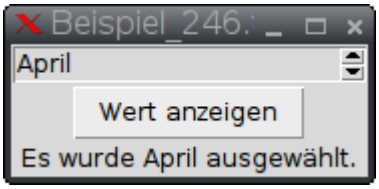
```
#!/usr/bin/env wish

ttk::spinbox .sbZahl -from 1 -to 5 -increment 1 -textvariable Zahl -state normal
pack .sbZahl
```





Listing 41.36: Spinbox auswerten (Beispiel246.tcl)

```
1  #!/usr/bin/env wish
2
3  set Liste {Januar Februar Maerz April Mai Juni}
4  set Auswahl ""
5
6  set Monat  [lindex $Liste 3]
7  ttk::spinbox .sbMonat -values $Liste -textvariable Monat -state readonly
8  ttk::button .btAnzeigen -text "Wert anzeigen" -command {set Auswahl "Es wurde $Monat ausgewaehlt."}
9  ttk::label .lbAuswahl -textvariable Auswahl
10
11 pack .sbMonat -side top
12 pack .btAnzeigen -side top
13 pack .lbAuswahl -side top
```



In Zeile 6 wird die Textvariable der Spinbox vorbelegt.

41.11 Listbox

Eine Listbox ist eine Liste mit Texteinträgen, aus denen der Anwender einen oder mehrere Einträge auswählen kann. Man kann die Listbox mit Scroll-Leisten ergänzen, falls nicht alle Einträge in die Listbox passen. Die Listbox gehört zu den klassischen Elementen und

hat nicht das Präfix `ttk::`. Das zugehörige ttk-Element ist das Treeview-Element, das man auch als Listbox verwenden kann.

Tabelle 41.14: Die wichtigsten Optionen

Option	Beschreibung
`-width AnzahlZeichen`	Breite
`-height AnzahlZeilen`	Höhe

`-selectmode single`
`-selectmode multiple`
`-selectmode browse`
`-selectmode extended`
	Selektionsmodus
	`browse` = genau 1 Eintrag, der sowohl per Maus-Klick als auch Cursor-Tasten ausgewählt werden kann

`single` = genau 1 Eintrag, der aber nur per Maus-Klick ausgewählt werden kann `multiple` = per Mausklick können mehrere Einträge selektiert werden `extended` = in Kombination mit der Strg-Taste oder Umschalt-Taste können mehrere Einträge selektiert werden. `-listvariable Liste`	Liste mit Einträgen
`-exportselection 0`	wenn mehrere Listboxen benutzt werden, bleibt durch diese Option die Seleketion jeder einzelnen Listbox bestehen
`-state normal` `-state disabled`	Status des Elements. `normal` = Element kann angeklickt werden `disabled` = Element kann nicht angeklickt werden
`-font Schrift`	Schrift
`-foreground Farbe`	Textfarbe
`-background Farbe`	Hintergrundfarbe

Tabelle 41.15: Die wichtigsten Aktionen

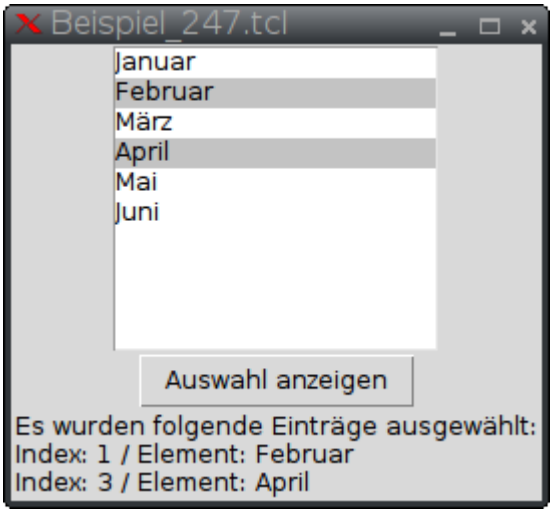
Aktion	Beschreibung
`listboxName curselection`	Liste mit den selektierten Einträgen
`listboxName selection set Von Bis`	Selektiert einen oder mehrere Einträge
`listboxName selection clear 0 end`	Hebt die Selektion auf
`listboxName see Index`	Bringt einen Eintrag in den sichtbaren Teil der Listbox

Tabelle 41.16: Virtuelles Ereignis

Ereignis	Beschreibung
`<<ListboxSelect>>`	der Anwender wählt einen Eintrag aus

Listing 41.37: Listbox, in der mehrere Einträge selektiert werden können (Beispiel247.tcl)

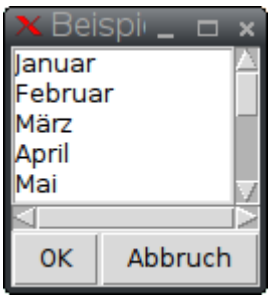
```
#!/usr/bin/env wish

proc AuswahlAnzeigen {Liste IndexAuswahl tmpText} {
  upvar $tmpText Text
  set Text "Es wurden folgende Eintraege ausgewaehlt:"
  foreach Index $IndexAuswahl {
    set Element [lindex $Liste $Index]
    append Text "\nIndex: $Index / Element: $Element"
  }
}
set Liste {Januar Februar Maerz April Mai Juni}
listbox .lbox -listvariable Liste -selectmode extended
button .bt -text "Auswahl anzeigen" -command {
   AuswahlAnzeigen $Liste [.lbox curselection] Text}
label .lb -textvariable Text -anchor w -justify left

pack .lbox -side top
pack .bt -side top
pack .lb -side bottom
```



In Zeile 13 wird mit dem Befehl `[.lbox curselection]` ermittelt, welche Einträge ausgewählt wurden. Der Befehl `curselection` gibt nicht direkt den Texteintrag wieder, sondern den Index des Eintrags. Wenn in der Listbox mehrere Einträge selektiert wurden, gibt der Befehl eine Liste zurück. In Zeile 4 wird ein Verweis auf die Variable `Text` im Hauptprogramm erzeugt. In Zeile 6 wird die Liste der Indices durchlaufen. In der Zeile 7 wird das zum Index gehörende Element aus der Liste ermittelt. In Zeile 8 wird der Index und das zugehörende Listenelement zur Variablen `Text` hinzugefügt.

Listing 41.38: Listbox mit Scroll-Leiste (Beispiel248.tcl)

```
1  #!/usr/bin/env wish
2
3  set Liste {Januar Februar Maerz April Mai Juni Juli August
       September Oktober November Dezember}
4
5  frame .fr
6  listbox .fr.lbox -width 5 -height 5 -listvariable Liste
       -xscrollcommand {.fr.sbX set} -yscrollcommand {.fr.sbY
       set}
7  scrollbar .fr.sbX -orient horizontal -command {.fr.lbox
       xview}
8  scrollbar .fr.sbY -command {.fr.lbox yview}
9
10 pack .fr.sbX -side bottom -fill x
11 pack .fr.sbY -side right -fill y
12 pack .fr.lbox -side top -expand yes -fill both
13
14 button .btOK -text OK
15 button .btAbbruch -text Abbruch
16
17 pack .fr -side top -expand yes -fill both
18 pack .btOK -side left
19 pack .btAbbruch -side right
```

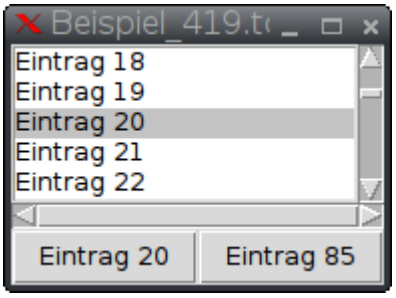


Die Listbox und die Scroll-Leisten werden gemeinsam in einen Rahmen gesetzt (Zeilen 10-12). In Zeile 17 wird dann der Rahmen mit den drei Elementen (Listbox und die beiden Scroll-Leisten) platziert.

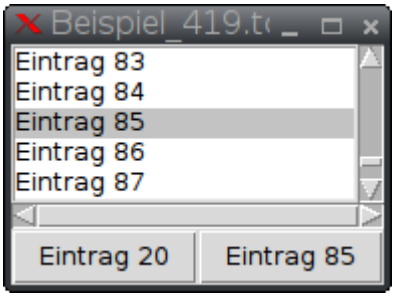
Listing 41.39: Selektion aufheben und einen Eintrag programmseitig auswählen (Beispiel419.tcl)

```
1  #!/usr/bin/env wish
2
3  proc EintragSelektieren {Index} {
4          .fr.lbox selection clear 0 end
5          .fr.lbox selection set $Index
6          .fr.lbox see $Index
7          update idletasks
8  }
9
10 for {set Index 0} {$Index < 100} {incr Index} {
11         lappend Liste "Eintrag $Index"
12 }
13
14 frame .fr
15 listbox .fr.lbox -width 5 -height 5 -listvariable Liste ↩
      -xscrollcommand {.fr.sbX set} -yscrollcommand {.fr.sbY ↩
      set}
16 scrollbar .fr.sbX -orient horizontal -command {.fr.lbox ↩
      xview}
17 scrollbar .fr.sbY -command {.fr.lbox yview}
18
19 pack .fr.sbX -side bottom -fill x
20 pack .fr.sbY -side right -fill y
21 pack .fr.lbox -side top -expand yes -fill both
22
23 button .bt1 -text "Eintrag 20" -command {↩
      EintragSelektieren 20}
24 button .bt2 -text "Eintrag 85" -command {↩
      EintragSelektieren 85}
25
26 pack .fr -side top -expand yes -fill both
27 pack .bt1 -side left
28 pack .bt2 -side right
```

Nach einem Klick auf den Button `Eintrag 20`:



Nach einem Klick auf den Button `Eintrag 85`:



InZeile 4 wird die bestehende Selektion aufgehoben. In Zeile 5 wird ein neuer Eintrag in der Listbox ausgewählt. In Zeile 6 wird der selektierte Eintrag in der Listbox angezeigt. Ohne diesen Befehl ist zwar ein Eintrag ausgewählt, aber der Anwender müsste erst mit Hilfe der Scroll-Leiste nach unten bzw. oben fahren, um den selektierten Eintrag zu sehen.

Die Listbox hat ein spezielles Ereignis `<<ListboxSelect>>`. Dadurch kann man einen Befehl oder eine Prozedur ausführen, wenn der Anwender einen Eintrag in der Listbox selektiert. Eine ausführliche Darstellung von Maus- und Tastaturereignissen werden in Kapitel 44 auf Seite 533 behandelt.

Listing 41.40: Das Ereignis «ListboxSelect» (Beispiel478.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Listbox {} {
4          set Index [.lb curselection]
5          puts "Index=$Index"
6  }
7
8  set Liste {a b c d e}
9
10 listbox .lb -listvariable Liste -selectmode single ;#
     browse ;#single
11 pack .lb -side left
12 bind .lb <<ListboxSelect>> {Listbox}
```

In Zeile 12 wird die Listbox mit dem Ereignis `<<ListboxSelect>>` verknüpft. Durch einen Klick auf den zweiten Eintrag in der Listbox wird das Ereignis ausgelöst und die Prozedur `Listbox` aufgerufen.

Wenn man mehrere Listboxen verwendet, entsteht zunächst ein Problem beim Ereignis `<<ListboxSelect>>`.

Listing 41.41: Problem beim Ereignis «ListboxSelect» bei zwei Listboxen (Beispiel479.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Listbox1 {} {
4          set Index [.lb1 curselection]
5          puts "Linke Listbox, Index=$Index"
6  }
7
8  proc Listbox2 {} {
9          set Index [.lb2 curselection]
10         puts "Rechte Listbox, Index=$Index"
11 }
12
13 set Liste1 {a b c d e}
14 set Liste2 {1 2 3 4 5}
15
16 listbox .lb1 -listvariable Liste1 -selectmode browse
17 listbox .lb2 -listvariable Liste2 -selectmode browse
18
19 pack .lb1 -side left
20 pack .lb2 -side left
21
22 bind .lb1 <<ListboxSelect>> {Listbox1}
23 bind .lb2 <<ListboxSelect>> {Listbox2}
```

Zunächst wird mit der Maus in der linken Listbox auf den zweiten Eintrag geklickt:

Dadurch wird das Ereignis `<<ListboxSelect>>` ausgelöst und die Prozedur zeigt den selektierten Eintrag an. Soweit ist alles in Ordnung. Wenn man danach in der rechten Listbox den fünften Eintrag anklickt, kommt es jedoch zu einem unerwarteten Ergebnis:

Obwohl der Mausklick in der rechten Listbox ausgeführt wurde, wird die Selektion in der linken Listbox aufgehoben und das Ereignis `<<ListboxSelect>>` der linken Listbox ausgelöst. Um dieses unerwartete Verhalten zu verhindern, setzt man bei beiden Listboxen die Option `-exportselection 0`.

Listing 41.42: Kein Problem beim Ereignis «ListboxSelect» bei zwei Listboxen durch die Option exportselection (Beispiel480.tcl)

```
#!/usr/bin/env wish

proc Listbox1 {} {
        set Index [.lb1 curselection]
```

```
5          puts "Linke Listbox, Index=$Index"
6  }
7
8  proc Listbox2 {} {
9          set Index [.lb2 curselection]
10         puts "Rechte Listbox, Index=$Index"
11 }
12
13 set Liste1 {a b c d e}
14 set Liste2 {1 2 3 4 5}
15
16 listbox .lb1 -listvariable Liste1 -selectmode browse ↩
      -exportselection 0
17 listbox .lb2 -listvariable Liste2 -selectmode browse ↩
      -exportselection 0
18
19 pack .lb1 -side left
20 pack .lb2 -side left
21
22 bind .lb1 <<ListboxSelect>> {Listbox1}
23 bind .lb2 <<ListboxSelect>> {Listbox2}
```

Auch dieses Mal wird zuerst mit der Maus in der linken Listbox auf den zweiten Eintrag geklickt:

Dadurch wird das Ereignis `<<ListboxSelect>>` ausgelöst und die Prozedur zeigt den selektierten Eintrag an. Anschließend wird in der rechten Listbox der fünfte Eintrag angeklickt.

Wie man sieht bleibt durch die Option `-exportselection 0` in den Zeilen 16 und 17 der Eintrag der linken Listbox weiterhin selektiert und es wird auch nicht das Ereignis `<<ListboxSelect>>` der linken Listbox ausgelöst.

41.12 Combobox

Eine Combobox ist ein Eingabe- und Auswahlfeld, das aufklappt und eine Liste an Auswahlmöglichkeiten anzeigt. Die Combobox gehört zu einer Reihe neu eingeführter Elemente in Tcl/Tk. Die Definition erfolgt deshalb über den Namensraum `ttk::combobox`.

Tabelle 41.17: Die wichtigsten Optionen

Option	Beschreibung
`-values Liste`	Liste der Einträge
`-textvariable Variable`	Speichert den ausgewählten Listeneintrag

`-state normal`
`-state readonly`
`-state disabled`
	Status der Combobox.
	`normal` = man kann eine beliebige Eingabe machen oder aus der Liste der vorgegebenen Werte auswählen
	`readonly` = man kann nur aus der Liste der vorgegebenen Werte auswählen
	`disabled` = das Element ist nicht aktiv

`-height`	Höhe der Liste, d.h. Anzahl der Listeneinträge, die unmittelbar (ohne zu scrollen) angezeigt werden

Tabelle 41.17: Die wichtigsten Optionen

Option	Beschreibung
`-width AnzahlZeichen`	Breite

`-justify left`
`-justify center`
`-justify right`
	Ausrichtung des Textes
	`left`=linksbündig
	`center`=zentriert
	`right`=rechtsbündig

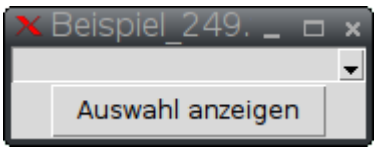
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil
`[Elementname current]`	Index des ausgewählten Elements

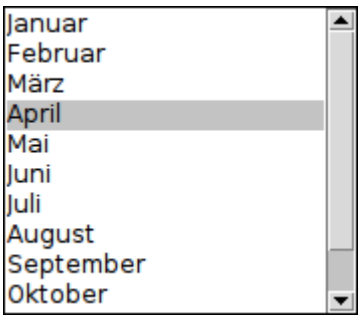
Tabelle 41.18: Virtuelles Ereignis

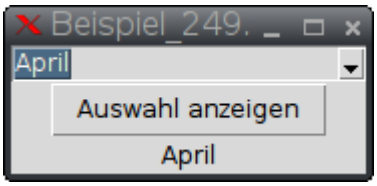
Ereignis	Beschreibung
`<<ComboboxSelected>>`	der Anwender wählt einen Eintrag aus

Listing 41.43: Auswahl aus der Liste (Beispiel249.tcl)

```
#!/usr/bin/env wish

set Liste {Januar Februar Maerz April Mai Juni Juli August September Oktober November Dezember}
set Auswahl [lindex $Liste 0]

ttk::combobox .cb -state readonly -values $Liste -textvariable Auswahl
ttk::button .bt -text "Auswahl anzeigen" -command {set Text $Auswahl}
ttk::label .lb -textvariable Text

pack .cb -side top
pack .bt -side top
pack .lb -side bottom
```



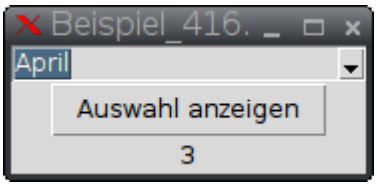




In Zeile 4 wird der erste Eintrag in der Combobox vorbelegt, um zu verhindern, dass der Anwender keine Auswahl trifft. Durch die Option `-state readonly` in Zeile 11, kann der Anwender nur Einträge aus der Liste auswählen. Er kann keinen eigenen Wert, der nicht in der Liste steht, eingeben.

Listing 41.44: Index des ausgewählten Elements (Beispiel416.tcl)

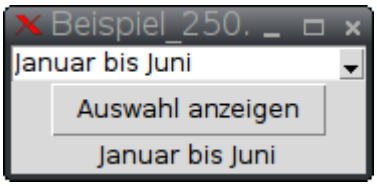
```
1  #!/usr/bin/env wish
2
3  set Liste {Januar Februar Maerz April Mai Juni Juli August
       September Oktober November Dezember}
4  set Auswahl [lindex $Liste 0]
5
6  ttk::combobox .cb -state readonly -values $Liste
     -textvariable Auswahl
7  ttk::button .bt -text "Auswahl anzeigen" -command {set
     Index [.cb current]}
8  ttk::label .lb -textvariable Index
9
10 pack .cb -side top
11 pack .bt -side top
12 pack .lb -side bottom
```



In Zeile 7 wird mit dem Befehl `[.cb current]` der Index des ausgewählten Elements ermittelt. Das erste Element hat den Index 0.

Listing 41.45: Beliebige Eingabe oder Auswahl aus der Liste (Beispiel250.tcl)

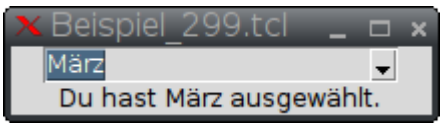
```
1  #!/usr/bin/env wish
2
3  set Liste {Januar Februar Maerz April Mai Juni Juli August
       September Oktober November Dezember}
4
5  ttk::combobox .cb -state normal -values $Liste
       -textvariable Auswahl
6  ttk::button .bt -text "Auswahl anzeigen" -command {set
       Text $Auswahl}
7  ttk::label .lb -textvariable Text
8
9  pack .cb -side top
10 pack .bt -side top
11 pack .lb -side bottom
```



Durch die Option `-state normal` in Zeile 10 kann der Anwender entweder einen Eintrag aus der Liste wählen oder er gibt einen eigenen Wert ein.

Listing 41.46: Bei der Auswahl aus der Liste wird automatisch eine Prozedur aufgerufen (Beispiel299.tcl)

```
#!/usr/bin/env wish

proc MeineProzedur {Auswahl} {
  set Text "Du hast $Auswahl ausgewaehlt."
  return $Text
}

set Liste {Januar Februar Maerz April Mai Juni Juli August
    September Oktober November Dezember}

ttk::combobox .cb -state readonly -values $Liste
    -textvariable Auswahl
ttk::label .lb -textvariable Text
pack .cb -side top
pack .lb -side bottom
bind .cb <<ComboboxSelected>> {set Text [MeineProzedur
    $Auswahl]}
```



Wenn der Benutzer einen Eintrag aus der Liste anklickt, wird automatisch die Prozedur

`MeineProzedur` aufgerufen. Dazu wird in Zeile 14 das Ereignis `<<ComboboxSelected>>` mit der Combobox verknüpft und festgelegt, welche Prozedur aufgerufen werden soll.

41.13 Scale

Das Scale-Element ist ein Schieberegler. Das klassische Scale-Element und das ttk-Scale-Element unterscheiden sich in den Optionen deutlich, weshalb im Folgenden beide Elemente betrachtet werden.

Tabelle 41.19: Die wichtigsten Optionen des klassischen Elements

Option	Beschreibung
`-orient horizontal` `-orient vertical`	Ausrichtung horizontal Ausrichtung vertikal
`-from Wert`	Untergrenze
`-to Wert`	Obergrenze
`-length Pixel`	Länge
`-variable Variable`	Speichert den eingestellten Wert
`-resolution Wert`	Schrittweite (Default-Wert = 1)
`-tickinterval Wert`	Skalierung (Beschriftung)
`-sliderlength Pixel`	Länge des Schiebereglers
`-showvalue 0` `-showvalue 1`	0 = Aktueller Wert wird nicht angezeigt 1 = Aktueller Wert wird angezeigt
`-label Text`	Beschriftung des Schiebereglers
`-font Schrift`	Schrift
`-foreground Farbe`	Textfarbe
`-background Farbe`	Hintergrundfarbe

Tabelle 41.20: Die wichtigsten Optionen des ttk-Elements

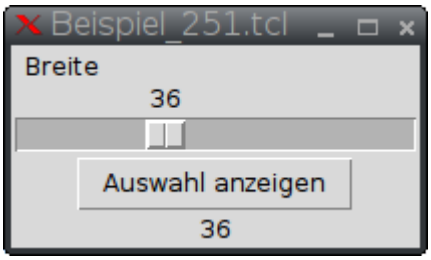
Option	Beschreibung
`-orient horizontal` `-orient vertical`	Ausrichtung horizontal Ausrichtung vertikal
`-from Wert`	Untergrenze
`-to Wert`	Obergrenze

Tabelle 41.20: Die wichtigsten Optionen des ttk-Elements

Option	Beschreibung
`-length Pixel`	Länge
`-variable Variable`	Speichert den eingestellten Wert
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil

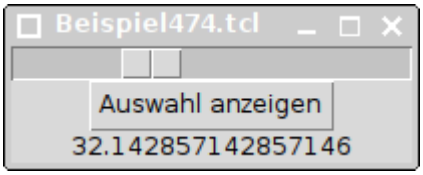
Listing 41.47: Klassischer Schieberegler ohne Skala (Beispiel251.tcl)

```
#!/usr/bin/env wish

scale .sc -orient horizontal -from 0 -to 100 -length 200 -sliderlength 20 -showvalue 1 -variable Wert -label "Breite"
ttk::button .bt -text "Auswahl anzeigen" -command {set Text $Wert}
ttk::label .lb -textvariable Text

pack .sc -side top
pack .bt -side top
pack .lb -side bottom
```



Listing 41.48: ttk-Schieberegler ohne Skala (Beispiel474.tcl)

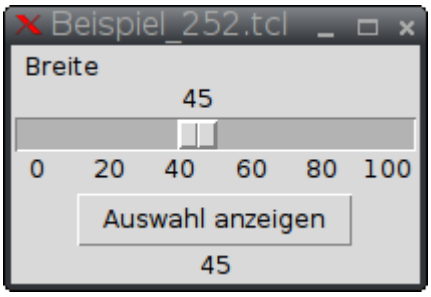
```
#!/usr/bin/env wish

ttk::scale .sc -orient horizontal -from 0 -to 100 -length 200 -variable Wert
ttk::button .bt -text "Auswahl anzeigen" -command {set Text $Wert}
ttk::label .lb -textvariable Text

pack .sc -side top
pack .bt -side top
pack .lb -side bottom
```



Beim ttk-Schieberegler kann man nicht die Position des Schiebereglers als Zahl oberhalb des Schiebereglers anzeigen.

Listing 41.49: Klassischer Schieberegler mit 20er-Skala und 5er-Schrittweite (Beispiel252.tcl)

```
#!/usr/bin/env wish

scale .sc -orient horizontal -from 0 -to 100 -resolution 5 -tickinterval 20 -length 200 -sliderlength 20 -showvalue 1 -variable Wert -label "Breite"
ttk::button .bt -text "Auswahl anzeigen" -command {set Text $Wert}
ttk::label .lb -textvariable Text

pack .sc -side top
pack .bt -side top
pack .lb -side bottom
```



Der ttk-Schieberegler bietet diese Möglichkeiten nicht.

41.14 Progressbar

Das Progressbar-Element ist eine Fortschrittsanzeige.

Tabelle 41.21: Die wichtigsten Optionen

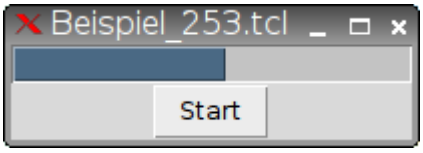
Option	Beschreibung
`-orient horizontal`	Ausrichtung horizontal
`-orient vertical`	Ausrichtung vertikal
`-maximum Wert`	Maximalwert

Tabelle 41.21: Die wichtigsten Optionen

Option	Beschreibung
`-length Pixel`	Länge
`-variable Variable`	Aktueller Wert, der aus einer Variablen stammt
`-value`	Aktueller Wert
`-mode indeterminate`	wenn der Maximalwert nicht bekannt ist
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil

Listing 41.50: Fortschrittsbalken (Beispiel253.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Schleife {Element} {
4    for {set Zaehler 0} {$Zaehler <= 100} {incr Zaehler} {
5      after 50
6      $Element configure -value $Zaehler
7      update idletask
8    }
9  }
10
11 set Wert 0
12 ttk::progressbar .pb -orient horizontal -maximum 100 ↩
     -length 200 -value 0
13 ttk::button .bt -text "Start" -command {Schleife .pb}
14 pack .pb -side top
15 pack .bt -side bottom
```

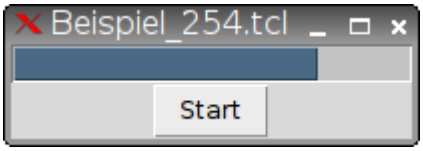


Die Prozedur `Schleife` in den Zeilen 3 bis 9 repräsentiert einen länger laufenden Vorgang, dessen Fortschritt durch den Fortschrittsbalken abgebildet wird. In Zeile 12 wird der Fortschrittsbalken erzeugt und der Wert `0` zugewiesen. In Zeile 6 wird der Wert des Fortschrittsbalken erhöht.

Listing 41.51: Fortschrittsbalken mit einer Variablen (Beispiel254.tcl)

```
1  #!/usr/bin/env wish
2
```

```
3  proc Schleife {tmp} {
4    upvar $tmp Zaehler
5    for {set Zaehler 0} {$Zaehler <= 100} {incr Zaehler} {
6      after 50
7      update idletask
8    }
9  }
10
11 set Wert 0
12 ttk::progressbar .pb -orient horizontal -maximum 100 ↲
     -length 200 -variable Wert
13 ttk::button .bt -text "Start" -command {Schleife Wert}
14 pack .pb -side top
15 pack .bt -side bottom
```



In Zeile 12 wird der Fortschrittsbalken mit der Variablen `Wert` verknüpft. In der Prozedur `Schleife` wird mit dem Befehl `upvar` auf die Variable `Wert` zugegriffen. In der Prozedur wird die Variable immer um eins erhöht.

Listing 41.52: Fortschrittsbalken mit unbekanntem Maximum (Beispiel255.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Schleife {Element} {
4    for {set Zaehler 0} {$Zaehler <= 100} {incr Zaehler} {
5      after 50
6      $Element configure -value $Zaehler
7      update idletask
8    }
9  }
10
11 set Wert 0
12 ttk::progressbar .pb -orient horizontal -mode ↲
     indeterminate -maximum 25 -length 200 -value 0
13 ttk::button .bt -text "Start" -command {Schleife .pb}
14 pack .pb -side top
15 pack .bt -side bottom
```

In Zeile 12 wird die Option `-mode indeterminate` hinzugefügt. Das macht man, wenn die Anzahl der Berechnungen (also der Maximalwert) unbekannt ist. Der Fortschrittsbalken wandert dann ständig von links nach rechts. Dabei definiert der Wert der Option

`-maximum` nach wie vielen Durchläufen der Balken einmal von links nach rechts gelaufen sein soll. In dem Beispiel wird die `for`-Schleife in Zeile 4 100-mal durchlaufen. Da das Maximum des Fortschrittsbalken auf 25 festgelegt wurde (Zeile 12), läuft der Balken insgesamt viermal von einer Seite auf die andere.

41.15 Notebook

Das Notebook-Element ist ein Container für andere Elemente. Es umfasst mehrere Reiter, die alternativ angewählt werden können.

Tabelle 41.22: Die wichtigsten Optionen

Option	Beschreibung
`add Frame -text Beschriftung`	Fügt dem Notebook ein Frame-Element hinzu und beschriftet den Reiter.
`select Frame`	Wählt ein Frame-Element aus.
`[Notebookname index current]`	Ermittelt den Index des aktiven Reiters. Der erste Reiter hat den Index 0.
`Notebookname tab Reitername -state disabled`	Deaktiviert den Reiter
`Notebookname tab Reitername -state normal`	Aktiviert den Reiter
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil

Tabelle 41.23: Virtuelles Ereignis

Ereignis	Beschreibung
`<<NotebookTabChanged>>`	der Anwender klickt auf einen Reiter

Listing 41.53: Notebook mit zwei Reitern (Beispiel256.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::notebook .n
4 ttk::frame .n.f1
5 ttk::frame .n.f2
6 .n add .n.f1 -text "Reiter 1"
```

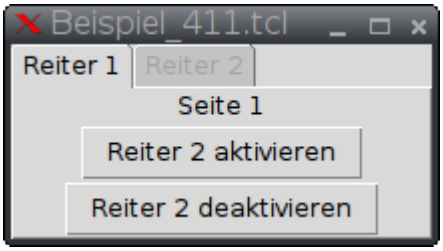
```
7  .n add .n.f2 -text "Reiter 2"
8
9  ttk::label .n.f1.l1 -text "Seite 1"
10 ttk::button .n.f1.b1 -text "OK"
11 ttk::label .n.f2.l1 -text "Seite 2"
12
13 pack .n -fill both -expand 1
14 pack .n.f1.l1 -side top
15 pack .n.f1.b1 -side bottom
16 pack .n.f2.l1 -side top
17
18 .n select .n.f1
```

In Zeile 3 wird das Notebook erzeugt. In den Zeilen 4 und 5 werden dem Notebook zwei Rahmen hinzugefügt. Diese sind Container für die weiteren Elemente. In den Zeilen 6 und 7 werden die beiden Rahmen dem Notebook hinzugefügt. In den Zeilen 9 bis 11 werden für jeden Rahmen weitere Elemente definiert. In den Zeilen 13 bis 16 werden die Elemente angezeigt. Die beiden Rahmen dürfen mit dem `pack`-Befehl nicht noch einmal aufgerufen werden, da sie bereits dem Notebook-Element hinzugefügt wurden. In der Zeile 18 wird der erste Rahmen selektiert.

Listing 41.54: Reiter im Notebook aktivieren und deaktivieren (Beispiel411.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::notebook .n
4  ttk::frame .n.f1
5  ttk::frame .n.f2
6  .n add .n.f1 -text "Reiter 1"
7  .n add .n.f2 -text "Reiter 2" -state disabled
8
9  ttk::label .n.f1.l1 -text "Seite 1"
10 ttk::button .n.f1.b1 -text "Reiter 2 aktivieren" -command ⤵
     {.n tab .n.f2 -state normal}
11 ttk::button .n.f1.b2 -text "Reiter 2 deaktivieren" ⤵
     -command {.n tab .n.f2 -state disabled}
12 ttk::label .n.f2.l1 -text "Seite 2"
```

```
13
14 pack .n -fill both -expand 1
15 pack .n.f1.l1 -side top
16 pack .n.f1.b1 -side top
17 pack .n.f1.b2 -side bottom
18 pack .n.f2.l1 -side top
19
20 .n select .n.f1
```



In Zeile 7 wird der `Reiter 2` deaktiviert. In Zeile 10 kann man durch Klick auf den Button den `Reiter 2` aktivieren. In Zeile 11 wird der Reiter wieder deaktiviert.
Das folgende Beispiel verwendet ein Mausklick-Ereignis (Klick mit der Maus auf einen Reiter), um den Index des aktiven Reiters anzuzeigen. Dies ist ein Vorgriff auf das Thema Tastatur- und Mausereignisse, das in Kapitel 44 auf Seite 533 genauer erklärt wird.

Listing 41.55: Ermittlung des aktiven Reiters (Beispiel412.tcl)

```
1  #!/usr/bin/env wish
2
3  proc MausklickNotebook {} {
4          switch [.n index current] {
5                  0 {puts "Es wurde auf Reiter 1 geklickt."}
6                  1 {puts "Es wurde auf Reiter 2 geklickt."}
7          }
8  }
9
10 ttk::notebook .n
11 ttk::frame .n.f1
12 ttk::frame .n.f2
13 .n add .n.f1 -text "Reiter 1"
14 .n add .n.f2 -text "Reiter 2"
15
16 ttk::label .n.f1.l1 -text "Seite 1"
17 ttk::button .n.f1.b1 -text "OK"
18 ttk::label .n.f2.l1 -text "Seite 2"
19
20 pack .n -fill both -expand 1
21 pack .n.f1.l1 -side top
22 pack .n.f1.b1 -side bottom
23 pack .n.f2.l1 -side top
24
25 .n select .n.f1
26
27 bind .n <ButtonRelease-1> {MausklickNotebook}
```

Wenn man auf einen Reiter 1 klickt, bekommt man folgende Ausgabe:

In Zeile 27 wird das Mausklick-Ereignis (siehe Kapitel 44 auf Seite 533) mit der Prozedur `MausklickNotebook` verknüpft. In Zeile 4 wird der aktive Reiter ermittelt. Der erste Reiter hat den Index 0.

41.16 Panedwindow

Das Panedwindow-Element ist ein Container, der mehrere Seiten nebeneinander darstellt. Zwischen den Seiten ist eine Trennlinie, die mit der Maus verschoben werden kann. Dadurch wird eine Seite breiter, die andere Seite schmäler.

Tabelle 41.24: Die wichtigsten Optionen

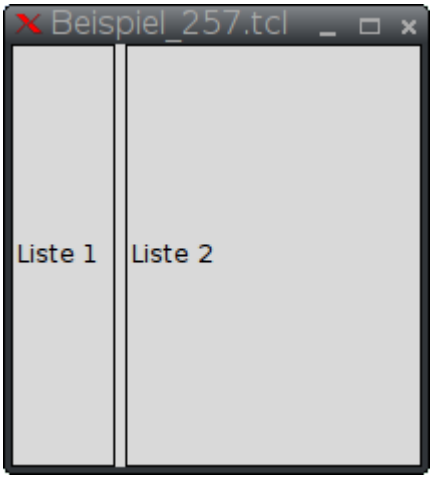
Option	Beschreibung
`add Frame`	Fügt dem Panedwindow ein Frame-Element hinzu.
`-orient horizontal` `-orient vertical`	Unterteilt das Panedwindow horizontal bzw. vertikal
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil

Listing 41.56: Panedwindow (Beispiel257.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::panedwindow .p -orient horizontal
4 ttk::frame .p.f1 -borderwidth 1 -relief solid
5 ttk::frame .p.f2 -borderwidth 1 -relief solid
6 .p add .p.f1
```

```
7  .p add .p.f2
8
9  ttk::label .p.f1.l1 -text "Liste 1" -anchor e -justify ↩
       left
10 ttk::label .p.f2.l1 -text "Liste 2" -anchor e -justify ↩
       left
11
12 pack .p -fill both -expand 1
13 pack .p.f1.l1 -side left
14 pack .p.f2.l1 -side left
```

Mit der Maus kann man die vertikale Trennlinie nach links und rechts schieben:

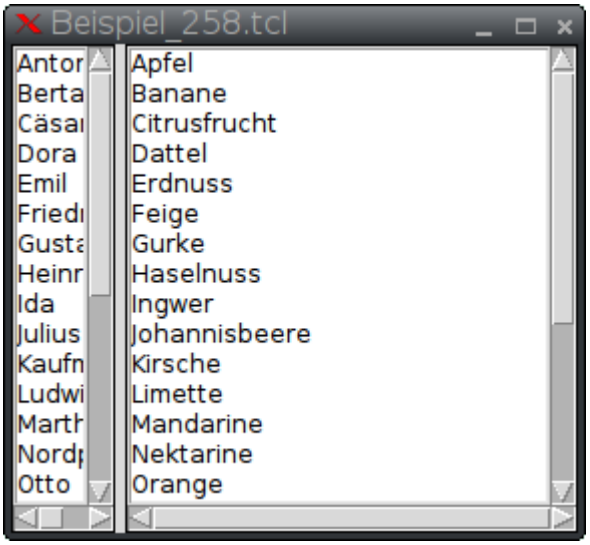


In Zeile 3 wird das Panedwindow definiert. In den Zeilen 4 und 5 werden zwei Rahmen erzeugt. In den Zeilen 6 und 7 werden die beiden Rahmen dem Panedwindow hinzugefügt. Man muss beachten, dass die Rahmen nicht mit dem Befehl `pack` platziert werden, sondern nur durch den `add`-Befehl (Zeilen 6 und 7).

Listing 41.57: Panedwindow mit zwei Listboxen und Scrollbars (Beispiel258.tcl)

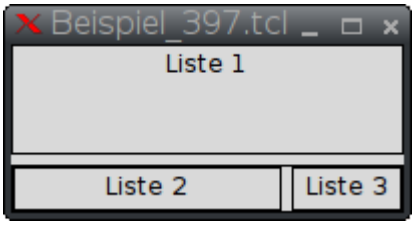
```
1  #!/usr/bin/env wish
2
3  set Liste1 {Anton Berta Caesar Dora Emil Friedrich Gustav Heinrich Ida Julius Kaufmann}
4  set Liste1 [concat $Liste1 {Ludwig Martha Nordpol Otto Paula Quelle Richard Samuel Schule Theodor}]
5  set Liste1 [concat $Liste1 {Ulrich Viktor Wilhelm Xanthippe Ypsilon Zacharias}]
6
7  set Liste2 {Apfel Banane Citrusfrucht Dattel Erdnuss Feige Gurke Haselnuss Ingwer Johannisbeere}
8  set Liste2 [concat $Liste2 {Kirsche Limette Mandarine Nektarine Orange Paprika Quitte Rosine}]
9  set Liste2 [concat $Liste2 {Stachelbeere Traube Ulgi Vanille Wassermelone Zitrone}]
10
11 ttk::panedwindow .p -orient horizontal
12 ttk::frame .p.f1 -borderwidth 1 -relief solid
13 ttk::frame .p.f2 -borderwidth 1 -relief solid
14 .p add .p.f1
15 .p add .p.f2
16
17 listbox .p.f1.lbox1 -width 15 -height 15 -xscrollcommand {.p.f1.sbX1 set} -yscrollcommand {.p.f1.sbY1 set} -listvariable Liste1
18 ttk::scrollbar .p.f1.sbX1 -orient horizontal -command {.p.f1.lbox1 xview}
19 ttk::scrollbar .p.f1.sbY1 -command {.p.f1.lbox1 yview}
20
21 listbox .p.f2.lbox1 -width 15 -height 15 -xscrollcommand {.p.f2.sbX1 set} -yscrollcommand {.p.f2.sbY1 set}
```

```
   -listvariable Liste2
ttk::scrollbar .p.f2.sbX1 -orient horizontal -command {.p.f2.lbox1 xview}
ttk::scrollbar .p.f2.sbY1 -command {.p.f2.lbox1 yview}

pack .p -fill both -expand 1
pack .p.f1.sbX1 -side bottom -fill x
pack .p.f1.sbY1 -side right -fill y
pack .p.f1.lbox1 -fill both -expand 1
pack .p.f2.sbX1 -side bottom -fill x
pack .p.f2.sbY1 -side right -fill y
pack .p.f2.lbox1 -fill both -expand 1
```



Listing 41.58: Panedwindow mit horizontaler und vertikaler Trennlinie (Beispiel397.tcl)

```
#!/usr/bin/env wish

ttk::panedwindow .pw1 -orient vertical
ttk::frame .pw1.fr1 -borderwidth 1 -relief solid
ttk::frame .pw1.fr2 -borderwidth 1 -relief solid
.pw1 add .pw1.fr1
.pw1 add .pw1.fr2
pack .pw1 -fill both -expand 1

ttk::panedwindow .pw1.fr2.pw2 -orient horizontal
ttk::frame .pw1.fr2.pw2.fr1 -borderwidth 1 -relief solid
ttk::frame .pw1.fr2.pw2.fr2 -borderwidth 1 -relief solid
.pw1.fr2.pw2 add .pw1.fr2.pw2.fr1
.pw1.fr2.pw2 add .pw1.fr2.pw2.fr2
pack .pw1.fr2.pw2 -fill both -expand 1

ttk::label .pw1.fr1.lb -text "Liste 1" -anchor e
pack .pw1.fr1.lb

ttk::label .pw1.fr2.pw2.fr1.lb -text "Liste 2" -anchor e
ttk::label .pw1.fr2.pw2.fr2.lb -text "Liste 3" -anchor e
pack .pw1.fr2.pw2.fr1.lb
pack .pw1.fr2.pw2.fr2.lb
```



In den Zeilen 3 bis 8 wird ein Panedwindow erzeugt, das vertikal unterteilt ist. In den oberen Bereich wird der Rahmen `.pw1.fr1` eingefügt, in den unteren Bereich der Rahmen `.pw1.fr2`. In den Zeilen 10 bis 15 wird in den unteren Rahmen ein weiteres Panedwindow eingefügt. Dieses Panedwindow ist horizontal unterteilt. In den linken Bereich wird der Rahmen `.pw1.fr2.pw2.fr1` eingefügt und in den rechten Bereich der Rahmen `.pw1.fr2.pw2.fr2`.

41.17 Treeview

Das Treeview-Element wird für eine spaltenweise oder eine baumartige Darstellung verwendet, wie z. B. bei einem Dateimanager, der Dateiname, Dateigröße, Dateidatum usw. spaltenweise anzeigt oder Ordner mit Dateien und Unterordner baumartig darstellt. Das Treeview-Element ersetzt außerdem das klassische Listbox-Element.

Das Treeview-Element enthält sogenannte Items, die jeweils eine Zeile mit Informationen enthält (z. B. Dateiname, Dateigröße, Dateidatum). Ein Item kann auch eine Verzweigung (Knoten) sein, der seinerseits eine Menge an Items umfasst.

Tabelle 41.25: Die wichtigsten Optionen

Option	Beschreibung
`-columns Liste`	Legt fest, wie viele Spalten eine Zeile hat. Dazu wird eine Liste mit Spaltennamen übergeben, die als interne Spaltennamen verwendet werden und zum Identifizieren der einzelnen Spalten dienen.
`-displaycolumns Liste`	Legt fest, welche Spalten (und in welcher Reihenfolge) angezeigt werden.
`-height Zeilen`	Legt fest, wie viele Zeilen angezeigt werden.
`-show headings`	Legt fest, dass die erste (interne) Spalte (mit dem Index 0) nicht dargestellt wird.

Tabelle 41.25: Die wichtigsten Optionen

Option	Beschreibung
`-selectmode browse` `-selectmode extended`	Legt fest, ob ein oder mehrere Elemente ausgewählt werden können. `browse` = es kann genau ein Element ausgewählt werden `extended` = es können mehrere Elemente (mittels der Strg-Taste) ausgewählt werden
`-cursor Cursor`	Mauszeiger-Symbol innerhalb des Elements
`-style Stil`	Stil
`.tv heading Spalte` `-text Text`	Beschriftet eine Spalte
`.tv heading Spalte` `-command {Befehle}`	Legt fest, welche Befehle ausgeführt werden, wenn man mit der Maus auf den Titel einer Spalte klickt
`.tv column Spalte` `-minwidth Pixel`	Mindestbreite einer Spalte in Pixel
`.tv column Spalte` `-stretch Boolean`	Legt fest, ob eine Spalte expandieren darf. Werte sind 0 (=nein) oder 1 (=ja)
`.tv insert {} end` `-values Element`	Fügt dem Wurzelknoten ein Element hinzu

`.tv insert {} end`
`-values Element`
`-tags {TagName}`
	Fügt dem Wurzelknoten ein Element hinzu und ordnet das Element dem `tag TagName` zu (über den `tag` erfolgt die Formatierung des Elements, z. B. die Schriftfarbe)

`.tv insert {} end` `-id ID -text Text`	Fügt einen Knoten unter dem Wurzelknoten ein
`.tv insert Elternknoten end` `-id Unterknoten -text Text`	Fügt einen Unterknoten unter dem Elternknoten ein
`.tv insert Knoten end` `-values Element`	Fügt dem Knoten ein Element hinzu
`.tv children Knoten`	Ermittelt alle Untereinträge des Knotens

Tabelle 41.25: Die wichtigsten Optionen

Option	Beschreibung
`[.tv selection]`	Gibt die ID der ausgewählten Elemente zurück.
`.tv selection set ID`	Markiert den Eintrag mit einer bestimmten ID
`.tv selection remove ID`	Hebt die Markierung für einen bestimmten Eintrag wieder auf
`[.tv item ID -value]`	Gibt den Eintrag mit einer bestimmten ID zurück
`.tv item ID -values Liste`	Ändert den Eintrag, in dem eine Liste mit den neuen Werten übergeben wird.
`.tv delete ID`	Löscht den Eintrag
`.tv see ID`	Macht den Eintrag sichtbar (d. h. verändert den Scrollbereich, so dass der Eintrag sichtbar ist)
`ttk::style configure Treeview.Heading -padding "0 0 0 0"`	Legt die Ränder der Titelzeile fest und damit zugleich die Höhe der Titelzeile (Reihenfolge ist links, oben, rechts, unten)
`.tv tag configure TagName -foreground Farbe`	Legt die Schriftfarbe für alle Elemente fest, die den `tag TagName` haben

Tabelle 41.26: Virtuelle Ereignisse

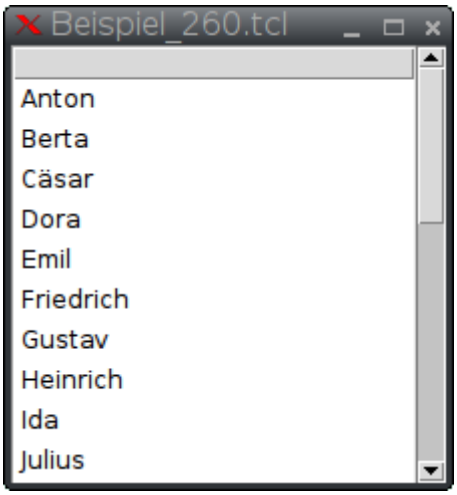
Ereignis	Beschreibung
`<<TreeviewSelect>>`	wenn die Selektion geändert wird
`<<TreeviewOpen>>`	unmittelbar bevor ein Knoten geöffnet wird
`<<TreeviewClose>>`	direkt nachdem ein Knoten geschlossen wurde

Bei den obigen Optionen kann die Spalte entweder über den Spaltennamen oder numerisch angesprochen werden. Wenn die Spalte numerisch angesprochen wird, gibt es zwei Varianten (siehe auch Beispiel 529):

- Spaltennummer mit einem vorangestellten # bezieht sich auf die angezeigten Spalten
- Spaltennummer ohne vorangestelltes# bezieht sich auf die Datenspalten

Listing 41.59: Treeview mit einer Spalte (Beispiel260.tcl)

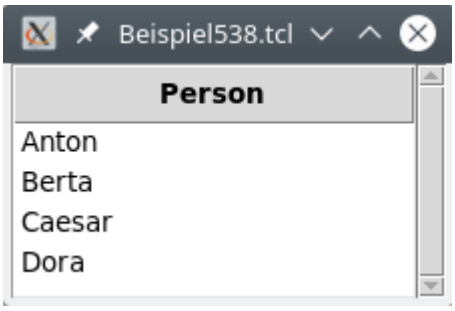
```
1  #!/usr/bin/env wish
2
3  set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav
     Heinrich Ida Julius Kaufmann}
4  set Liste [concat $Liste {Ludwig Martha Nordpol Otto Paula
     Quelle Richard Samuel Theodor}]
5  set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe
     Ypsilon Zacharias}]
6
7  ttk::treeview .tv -columns Person -show headings
     -yscrollcommand {.sbY set}
8  foreach Element $Liste {
9     .tv insert {} end -values $Element
10 }
11 ttk::scrollbar .sbY -command {.tv yview}
12
13 pack .sbY -side right -fill y
14 pack .tv -side left
```



In Zeile 7 wird der Treeview definiert. Die Option `-columns` legt die Spalten fest. In diesem Beispiel gibt es nur die Spalte `Person`. Wie man in der Ausgabe sieht, hat die Spalte noch keine Überschrift. In Zeile 9 werden zeilenweise die Daten in den Treeview eingetragen. `.tv` ist der Elementname. Die geschweifte Klammer `{}` legt fest, dass die Daten unterhalb des Wurzelknotens eingefügt werden. `end` legt die Position fest, also nach dem letzten Element. Die Option `-values $Element` enthält die Daten. Dabei ist die Variable `Element` eine Liste, in diesem Fall nur mit einem Wert (weil der Treeview einspaltig ist), sonst aber oft mit mehreren Werten.

Listing 41.60: Treeview mit Spaltentitel (Beispiel538.tcl)

```
1  #!/usr/bin/env wish
2
3  set Liste {Anton Berta Caesar Dora}
4
5  ttk::treeview .tv -columns Person -show headings ↩
       -yscrollcommand {.sbY set}
6  .tv heading Person -text "Person"
7  ttk::style configure Treeview.Heading -padding "0 5 0 5"
8
9  foreach Element $Liste {
10     .tv insert {} end -values $Element
11 }
12 ttk::scrollbar .sbY -command {.tv yview}
13
14 pack .sbY -side right -fill y
15 pack .tv -side left
```



In Zeile 6 wird der Spaltentitel definiert und in Zeile 7 dessen Ränder. Dadurch legt man zugleich die Höhe der Titelzeile fest. Die Reihenfolge der Parameter bei der `padding`-Option ist links, oben, rechts und unten.

Listing 41.61: Elemente einfügen ohne list-Befehl (Beispiel533.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::treeview .tv -columns Person -show headings ↩
       -yscrollcommand {.sbY set}
4  .tv heading Person -text "Person"
5  .tv insert {} end -values "Anton Maier"
6  .tv insert {} end -values "Berta Schmidt"
7  ttk::scrollbar .sbY -command {.tv yview}
8
9  pack .sbY -side right -fill y
10 pack .tv -side left
```

Der `insert`-Befehl erwartet eine Liste mit Elementen. Wenn die einzelnen Elemente Leerzeichen enthalten (Zeilen 5 und 6), muss man darauf achten, den `list`-Befehl zu verwenden (siehe folgendes Beispiel 534). Sonst wird nur der Textteil bis zum Leerzeichen eingefügt.

Listing 41.62: Elemente einfügen mit list-Befehl (Beispiel534.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::treeview .tv -columns Person -show headings ↲
     -yscrollcommand {.sbY set}
4  .tv heading Person -text "Person"
5  .tv insert {} end -values [list "Anton Maier"]
6  .tv insert {} end -values [list "Berta Schmidt"]
7  ttk::scrollbar .sbY -command {.tv yview}
8
9  pack .sbY -side right -fill y
10 pack .tv -side left
```

In den Zeilen 5 und 6 wurden die Elemente mit Hilfe des `list`-Befehls eingefügt. Dadurch wird das Leerzeichen im Text nicht als Spaltentrennung interpretiert.

Listing 41.63: Elemente formatieren (Beispiel535.tcl)

```
1  #!/usr/bin/env wish
2
3  set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav ↲
     Heinrich Ida Julius Kaufmann}
4  set Liste [concat $Liste {Ludwig Martha Nordpol Otto Paula↲
      Quelle Richard Samuel Theodor}]
5  set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe ↲
     Ypsilon Zacharias}]
6
7  ttk::treeview .tv -columns Person -show headings ↲
     -yscrollcommand {.sbY set}
8  .tv heading Person -text "Person"
9  set i 0
10 foreach Element $Liste {
11   if {[expr $i % 2] == 0} {
12     .tv insert {} end -values $Element -tags {TreeviewRot}
13   } else {
14     .tv insert {} end -values $Element -tags {TreeviewBlau↲
         }
15   }
16   incr i
17 }
```

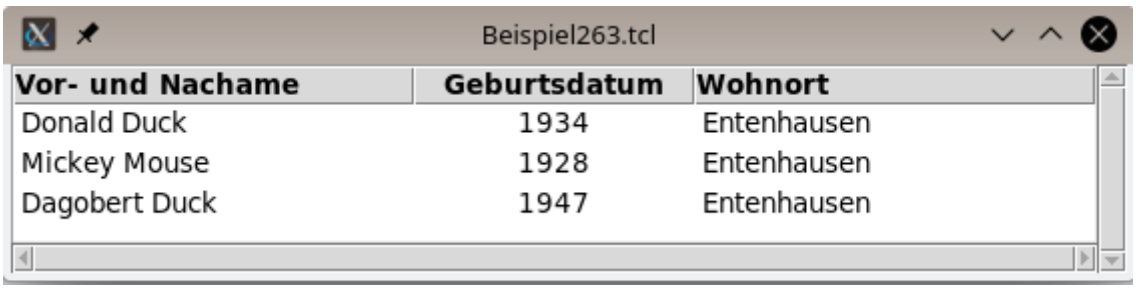
```
18 ttk::scrollbar .sbY -command {.tv yview}
19
20 .tv tag configure TreeviewRot -foreground red
21 .tv tag configure TreeviewBlau -foreground blue
22
23 pack .sbY -side right -fill y
24 pack .tv -side left
```

Ab der Tcl/Tk Version 8.6.10 kann man die Elemente unterschiedlich formatieren, z. B. unterschiedliche Schriftfarben geben. Dazu ordnet man das Element beim Einfügen in den Treeview einem sogenannten `tag` zu (Zeilen 12 und 14). Die einzelnen Tags können individuell formatiert werden (Zeilen 20 und 21). Wenn man ein Element mehreren `tags` zuordnen will, dann werden die Tag-Namen als Liste angegeben, z. B. `-tags {TagName1 TagName2 TagName3}`.

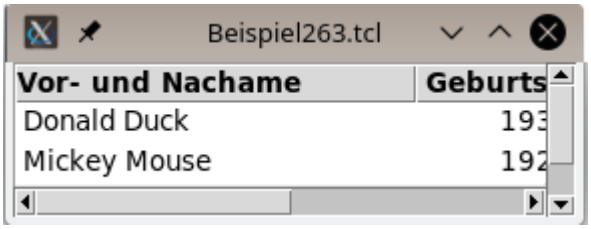
Listing 41.64: Treeview mit mehreren Spalten, Scroll-Leisten und Formatierung (Beispiel263.tcl)

```
1  #!/usr/bin/env wish
2
3  set ListeZeile1 ""
4  lappend ListeZeile1 "1"
5  lappend ListeZeile1 "Donald Duck"
6  lappend ListeZeile1 "1934"
7  lappend ListeZeile1 "Entenhausen"
8
9  set ListeZeile2 ""
10 lappend ListeZeile2 "2"
11 lappend ListeZeile2 "Mickey Mouse"
12 lappend ListeZeile2 "1928"
13 lappend ListeZeile2 "Entenhausen"
14
15 set ListeZeile3 ""
16 lappend ListeZeile3 "3"
17 lappend ListeZeile3 "Dagobert Duck"
18 lappend ListeZeile3 "1947"
19 lappend ListeZeile3 "Entenhausen"
```

```
20
21 set ListeEintraege ""
22 lappend ListeEintraege $ListeZeile1
23 lappend ListeEintraege $ListeZeile2
24 lappend ListeEintraege $ListeZeile3
25
26 ttk::treeview .tv -columns {ID Name Datum Ort} ↩
     -displaycolumns {Name Datum Ort} -show headings ↩
     -yscrollcommand {.sbY set} -xscrollcommand {.sbX set}
27 .tv column Name -minwidth 200 -stretch 1
28 .tv column Datum -minwidth 120 -stretch 1
29 .tv column Ort -minwidth 200 -stretch 1
30
31 foreach Zeile $ListeEintraege {
32   .tv insert {} end -values $Zeile
33 }
34
35 .tv heading Name -text "Vor- und Nachame"
36 .tv heading Datum -text "Geburtsdatum"
37 .tv heading Ort -text "Wohnort"
38
39 .tv heading Name -anchor w
40 .tv heading Datum -anchor center
41 .tv heading Ort -anchor w
42
43 .tv column Name -anchor w
44 .tv column Datum -anchor center
45 .tv column Ort -anchor w
46
47 .tv column Datum -width 140
48
49 ttk::scrollbar .sbY -command {.tv yview}
50 ttk::scrollbar .sbX -orient horizontal -command {.tv xview↩
     }
51
52 pack .sbY -side right -fill y
53 pack .sbX -side bottom -fill x
54 pack .tv -side left -expand yes -fill both
```



Wenn man das Fenster kleiner macht:

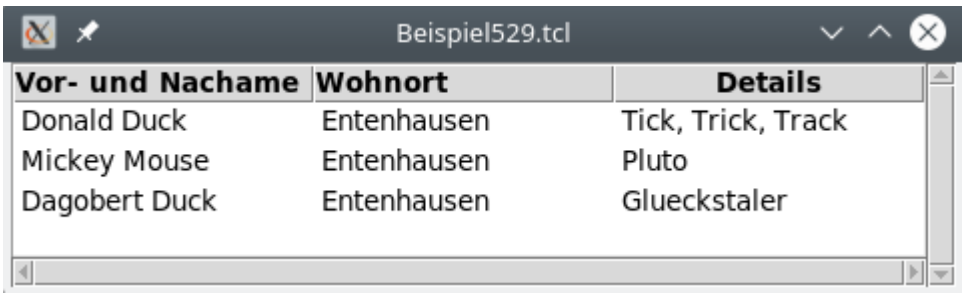


Jede Zeile des Treeviews besteht aus einer Liste mit den Spalteninhalten. Der gesamte Treeview besteht aus einer Liste mit den einzelnen Zeilen-Listen (es handelt sich somit um Listen in einer Liste). In den Zeilen 3 bis 7 werden die Einträge der ersten Zeile festgelegt. Es handelt sich dabei um eine Liste mit vier Elementen. In den Zeilen 9 bis 13 wird eine weitere Liste für die zweite Zeile festgelegt, und in den Zeilen 15 bis 19 wird eine dritte Zeile definiert. In den Zeilen 21 bis 24 werden die Zeilen-Listen zur Treeview-Liste zusammengefasst (Listen in Liste). In Zeile 26 wird der Treeview definiert. Die Optionen bedeuten:
Die Option `-columns` legt die (internen) Spaltenname fest.
Die Option `-displaycolumns` bestimmt, welche Spalten angezeigt werden sollen. In dem Beispiel soll die Spalte `ID` nicht angezeigt werden.
Die Option `-show headings` legt fest, dass Spaltenüberschriften angezeigt werden sollen.
In den Zeilen 27 bis 29 wird die Mindestbreite der sichtbaren Spalten festgelegt. Außerdem wird definiert, dass die Zeilen in der Breite expandieren dürfen. Anstelle der Spaltennamen kann man auch die Spaltennummern verwenden. Die erste Spalte hat die Nummer 0. Somit ist die Spalte `Name` die Nummer 1.
In den Zeilen 31 bis 33 werden die einzelnen Zeilen dem Treeview hinzugefügt. In den Zeilen 35 bis 37 werden die Spaltenüberschriften festgelegt. In den Zeilen 39 bis 41 wird die Ausrichtung (w=West, e=East (Ost), center=zentriert) der Spaltenüberschriften festgelegt. Die Befehle beeinflussen aber nicht die Ausrichtung der Spalteninhalte. Die wird in den Zeilen 43 bis 45 festgelegt. In Zeile 47 wird die Breite der Spalte `Datum` definiert.

Listing 41.65: Spalten numerisch oder über Name ansprechen (Beispiel529.tcl)

```
1  #!/usr/bin/env wish
2
3  set ListeZeile1 ""
4  lappend ListeZeile1 "1"
5  lappend ListeZeile1 "Donald Duck"
6  lappend ListeZeile1 "1934"
7  lappend ListeZeile1 "Entenhausen"
8  lappend ListeZeile1 "Tick, Trick, Track"
9
10 set ListeZeile2 ""
11 lappend ListeZeile2 "2"
12 lappend ListeZeile2 "Mickey Mouse"
13 lappend ListeZeile2 "1928"
14 lappend ListeZeile2 "Entenhausen"
15 lappend ListeZeile2 "Pluto"
16
17 set ListeZeile3 ""
18 lappend ListeZeile3 "3"
```

```
19 lappend ListeZeile3 "Dagobert Duck"
20 lappend ListeZeile3 "1947"
21 lappend ListeZeile3 "Entenhausen"
22 lappend ListeZeile3 "Glueckstaler"
23
24 set ListeEintraege ""
25 lappend ListeEintraege $ListeZeile1
26 lappend ListeEintraege $ListeZeile2
27 lappend ListeEintraege $ListeZeile3
28
29 ttk::treeview .tv -columns {ID Name Datum Ort Sonstiges} ↩
      -displaycolumns {Name Ort Sonstiges} -show headings ↩
      -yscrollcommand {.sbY set} -xscrollcommand {.sbX set}
30 .tv column Name -minwidth 150 -stretch 1
31 .tv column Datum -minwidth 100 -stretch 1
32 .tv column Ort -minwidth 150 -stretch 1
33
34 foreach Zeile $ListeEintraege {
35   .tv insert {} end -values $Zeile
36 }
37
38 .tv heading Name -text "Vor- und Nachame"
39 .tv heading Datum -text "Geburtsdatum"
40 .tv heading 3 -text "Wohnort"
41 .tv heading #3 -text "Details"
42
43 .tv heading Name -anchor w
44 .tv heading Datum -anchor center
45 .tv heading Ort -anchor w
46
47 .tv column Name -anchor w
48 .tv column Datum -anchor center
49 .tv column Ort -anchor w
50
51 .tv column Datum -width 140
52
53 ttk::scrollbar .sbY -command {.tv yview}
54 ttk::scrollbar .sbX -orient horizontal -command {.tv xview↩
      }
55
56 pack .sbY -side right -fill y
57 pack .sbX -side bottom -fill x
58 pack .tv -side left -expand yes -fill both
```

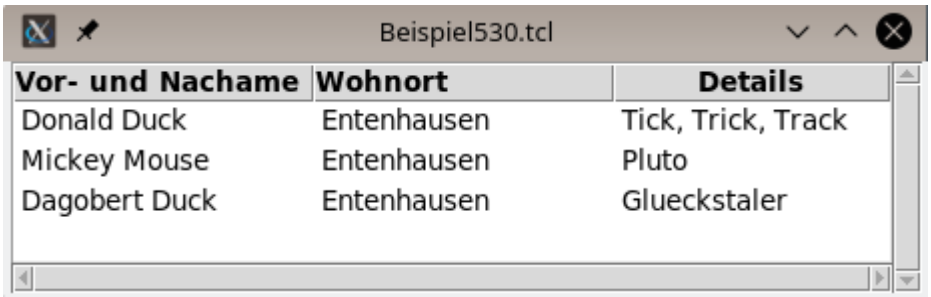


In Zeile 29 werden mit der Option `-columns` die fünf Datenspalten `ID`, `Name`, `Datum`, `Ort` und `Sonstiges` festgelegt. Mit der Option `-displaycolumns` weden die Spalten ausgewählt, die angezeigt werden: `Name`, `Ort` und `Sonstiges`. In den Zeilen 38 und 39 werden die Spaltentitel anhand des Spaltennamens gesetzt, in der Zeile 40 über die Datenspalte (Datenspalte `3` ist das `Datum`) und in Zeile 41 über die Displayspalte (Displayspalte `#3` ist `Sonstiges`) . Beachten Sie, dass man ein `#` Zeichen der Spalte voranstellen muss, wenn man die Displayspalte meint.
Man kann die Titelzeile auch dazu verwenden, dass ein Befehl ausgeführt wird, wenn man mit der Maus auf einen Spaltentitel klickt. Zum Beispiel könnte man damit den Inhalt des Treeviews auf- und absteigend sortieren.

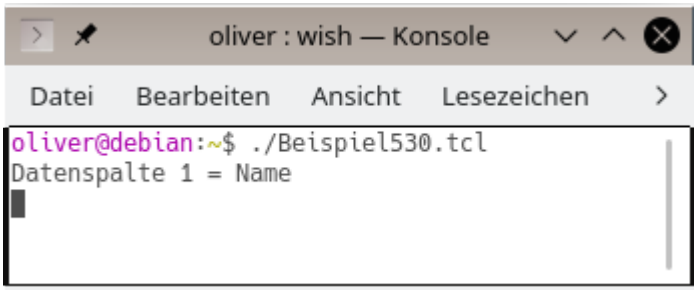
Listing 41.66: Spaltentitel mit Befehlen verknüpfen (Beispiel530.tcl)

```
1  #!/usr/bin/env wish
2
3  set ListeZeile1 ""
4  lappend ListeZeile1 "1"
5  lappend ListeZeile1 "Donald Duck"
6  lappend ListeZeile1 "1934"
7  lappend ListeZeile1 "Entenhausen"
8  lappend ListeZeile1 "Tick, Trick, Track"
9
10 set ListeZeile2 ""
11 lappend ListeZeile2 "2"
12 lappend ListeZeile2 "Mickey Mouse"
13 lappend ListeZeile2 "1928"
14 lappend ListeZeile2 "Entenhausen"
15 lappend ListeZeile2 "Pluto"
16
17 set ListeZeile3 ""
18 lappend ListeZeile3 "3"
19 lappend ListeZeile3 "Dagobert Duck"
20 lappend ListeZeile3 "1947"
21 lappend ListeZeile3 "Entenhausen"
22 lappend ListeZeile3 "Glueckstaler"
23
24 set ListeEintraege ""
25 lappend ListeEintraege $ListeZeile1
26 lappend ListeEintraege $ListeZeile2
27 lappend ListeEintraege $ListeZeile3
28
29 ttk::treeview .tv -columns {ID Name Datum Ort Sonstiges} ↩
     -displaycolumns {Name Ort Sonstiges} -show headings ↩
```

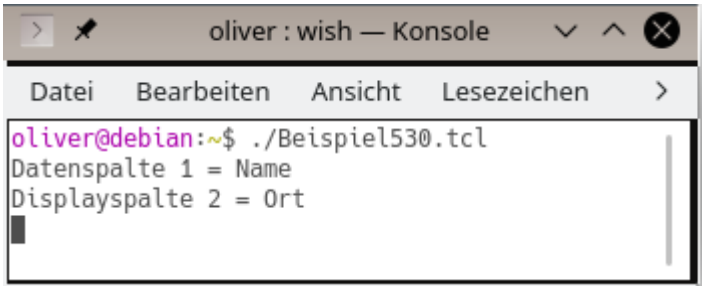
```
   -yscrollcommand {.sbY set} -xscrollcommand {.sbX set}
.tv column Name -minwidth 150 -stretch 1
.tv column Datum -minwidth 100 -stretch 1
.tv column Ort -minwidth 150 -stretch 1

foreach Zeile $ListeEintraege {
  .tv insert {} end -values $Zeile
}

.tv heading 1 -command {puts "Datenspalte 1 = Name"}
.tv heading 2 -command {puts "Datenspalte 2 = Datum"}
.tv heading #2 -command {puts "Displayspalte 2 = Ort"}

.tv heading Name -text "Vor- und Nachame"
.tv heading Datum -text "Geburtsdatum"
.tv heading 3 -text "Wohnort"
.tv heading #3 -text "Details"

.tv heading Name -anchor w
.tv heading Datum -anchor center
.tv heading Ort -anchor w

.tv column Name -anchor w
.tv column Datum -anchor center
.tv column Ort -anchor w

.tv column Datum -width 140

ttk::scrollbar .sbY -command {.tv yview}
ttk::scrollbar .sbX -orient horizontal -command {.tv xview}

pack .sbY -side right -fill y
pack .sbX -side bottom -fill x
pack .tv -side left -expand yes -fill both
```



Nach einem Mausklick auf den Titel der ersten Spalte:



Nach einem Mausklick auf den Titel der zweiten Spalte:

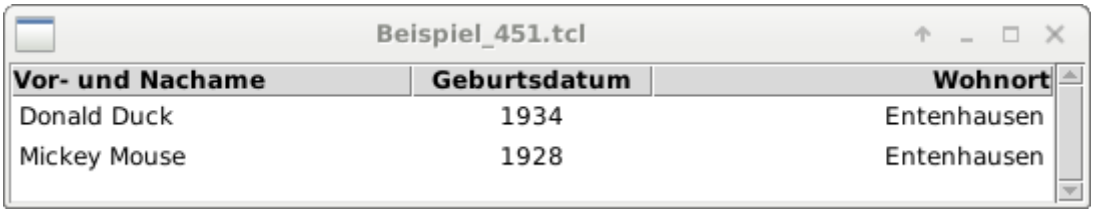


In den Zeilen 38 bis 40 werden drei Spaltentitel mit einem Befehl verknüpft, wenn man mit der Maus auf den Titel klickt. In den Zeilen 38 und 39 werden die Datenspalten adressiert, in Zeile 40 wird die Displayspalte angesprochen.

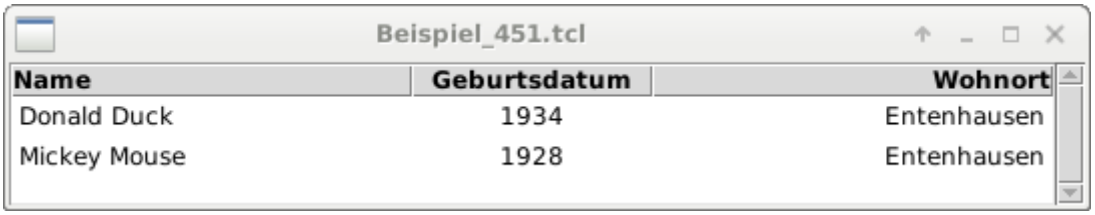
Listing 41.67: Spaltenüberschrift nachträglich ändern (Beispiel451.tcl)

```
1  #!/usr/bin/env wish
2
3  set ListeZeile1 ""
4  lappend ListeZeile1 "1"
5  lappend ListeZeile1 "Donald Duck"
6  lappend ListeZeile1 "1934"
7  lappend ListeZeile1 "Entenhausen"
8
9  set ListeZeile2 ""
10 lappend ListeZeile2 "2"
11 lappend ListeZeile2 "Mickey Mouse"
12 lappend ListeZeile2 "1928"
13 lappend ListeZeile2 "Entenhausen"
14
15 set ListeEintraege ""
16 lappend ListeEintraege $ListeZeile1
17 lappend ListeEintraege $ListeZeile2
18
19 ttk::treeview .tv -columns {ID Name Datum Ort} ↩
      -displaycolumns {Name Datum Ort} -show headings ↩
      -yscrollcommand {.sbY set}
20
21 foreach Zeile $ListeEintraege {
22   .tv insert {} end -values $Zeile
23 }
```

```
24
25 .tv heading Name -text "Vor- und Nachame"
26 .tv heading Datum -text "Geburtsdatum"
27 .tv heading Ort -text "Wohnort"
28
29 .tv heading Name -anchor w
30 .tv heading Datum -anchor center
31 .tv heading Ort -anchor e
32
33 .tv column Datum -width 120
34
35 .tv column Name -anchor w
36 .tv column Datum -anchor center
37 .tv column Ort -anchor e
38
39 ttk::scrollbar .sbY -command {.tv yview}
40
41 pack .sbY -side right -fill y
42 pack .tv -side left
43
44 update idletasks
45 after 2000
46
47 .tv heading Name -text "Name"
48 update idletasks
```



Und nach zwei Sekunden:



In Zeile 47 wird die Überschrift der Spalte `Name` geändert.

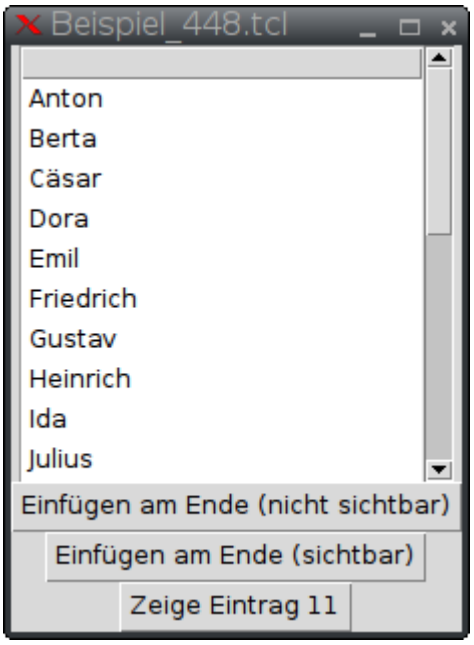
Listing 41.68: Treeview-Eintrag mit eigener ID hinzufügen und anzeigen (Scrollbereich verschieben) (Beispiel448.tcl)

```
1 #!/usr/bin/env wish
2
3 set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav ↲
    Heinrich Ida Julius Kaufmann}
```

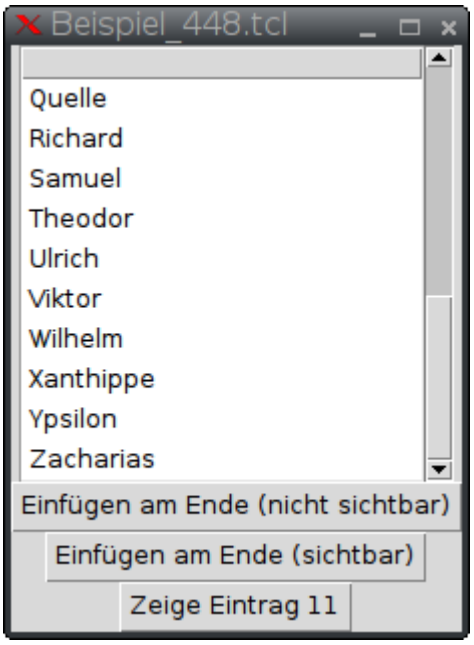
```
4  set Liste [concat $Liste {Ludwig Martha Nordpol Otto Paula
       Quelle Richard Samuel Theodor}]
5  set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe
     }]
6
7  ttk::frame .fr
8  ttk::treeview .fr.tv -columns Person -show headings
     -yscrollcommand {.fr.sbY set}
9  set ID 0
10 foreach Element $Liste {
11     .fr.tv insert {} end -id $ID -values $Element
12     incr ID
13 }
14 ttk::scrollbar .fr.sbY -command {.fr.tv yview}
15
16 ttk::button .bt1 -text "Einfuegen am Ende (nicht sichtbar)
     " -command {.fr.tv insert {} end -id 31 -values "
     Ypsilon"}
17 ttk::button .bt2 -text "Einfuegen am Ende (sichtbar)"
     -command {.fr.tv insert {} end -id 32 -values "
     Zacharias"; .fr.tv see 32}
18 ttk::button .bt3 -text "Zeige Eintrag 11" -command {.fr.tv
       see 10}
19
20 pack .fr.sbY -side right -fill y
21 pack .fr.tv -side left
22
23 pack .fr -side top
24 pack .bt1 -side top
25 pack .bt2 -side top
26 pack .bt3 -side top
```

Nach dem Programmstart:

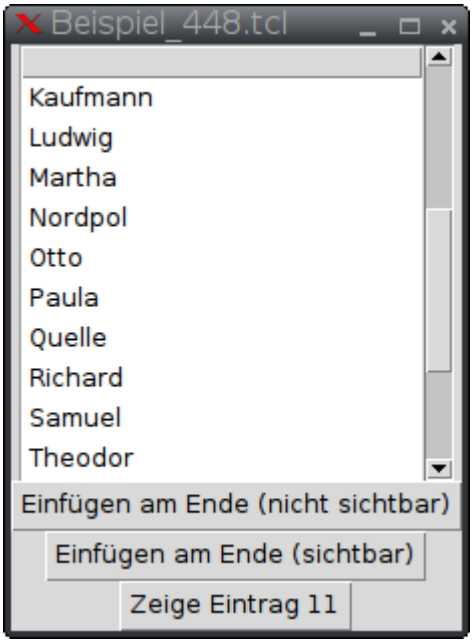
Nach dem Klick auf den Button `Einfügen am Ende (nicht sichtbar)`:



Nach dem Klick auf den Button `Einfügen am Ende (sichtbar)`:



Nach dem Klick auf den Button `Zeige Eintrag 11`:

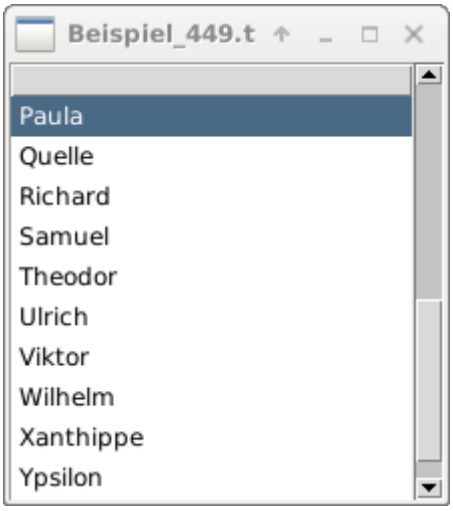


Jeder Eintrag im Treeview hat eine eindeutige ID. Diese ID muss entweder mit der Option `-id` mitgegeben werden oder sie wird automatisch erzeugt (als Rückgabewert des `insert`-Befehls). In Zeile 11 werden die Namen dem Treeview hinzugefügt. Dabei erhält jeder Eintrag durch die Option `-id` eine eindeutige ID zugewiesen. Über diese ID kann man später auf den Eintrag zugreifen. In Zeile 16 wird der Eintrag `Ypsilon` am Ende des Treeviews hinzugefügt. Er erhält die ID `31`. Allerdings rückt der Eintrag nicht in den sichtbaren Bereich. In Zeile 17 wird der Eintrag `Zacharias` am Ende hinzugefügt und erhält die ID `32`. Durch den Befehl `.fr.tv see 32` wird der Eintrag in den sichtbaren Bereich des Treeview verschoben. In Zeile 18 wird mit dem Befehl `.fr.tv see 10` der 11. Eintrag (er hat die ID 10) in den sichtbaren Bereich verschoben.

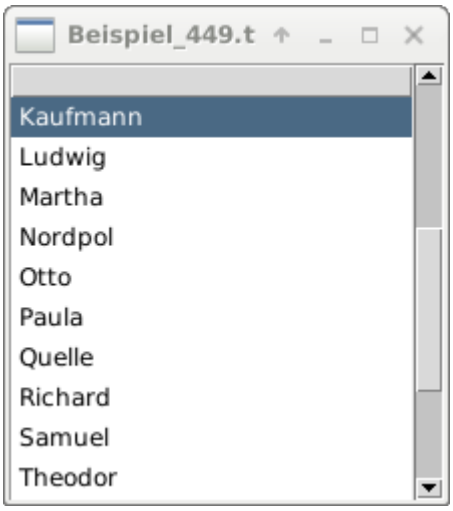
Listing 41.69: Eintrag programmseitig selektieren (Beispiel449.tcl)

```
1  #!/usr/bin/env wish
2
3  set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav
     Heinrich Ida Julius Kaufmann}
4  set Liste [concat $Liste {Ludwig Martha Nordpol Otto Paula
     Quelle Richard Samuel Theodor}]
5  set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe
     Ypsilon Zacharias}]
6
7  ttk::treeview .tv -selectmode browse -columns Person -show
     headings -yscrollcommand {.sbY set}
8
9  set ID 0
10 foreach Element $Liste {
11     .tv insert {} end -id $ID -values $Element
12     incr ID
13 }
14 ttk::scrollbar .sbY -command {.tv yview}
15
```

```
16 pack .sbY -side right -fill y
17 pack .tv -side left
18
19 .tv selection set 15
20 .tv see 15
21 update idletasks
22
23 after 2000
24
25 .tv selection remove 15
26 .tv selection set 10
27 .tv see 10
28 update idletasks
```



Und nach zwei Sekunden:

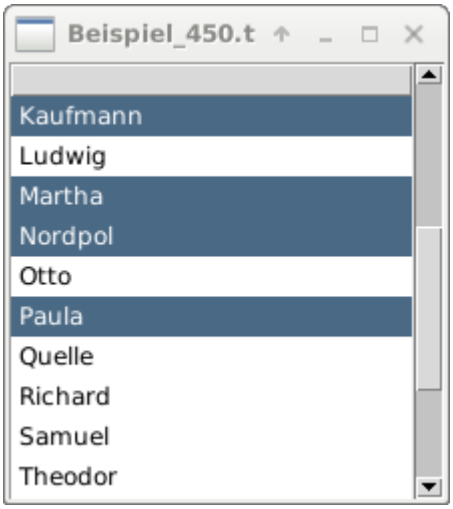


In den Zeilen 9 bis 13 werden die einzelnen Listenelemente in den Treeview eingetragen. Dabei wird die Variable `ID` hochgezählt und dem Treeview-Eintrag zugeordnet. Dadurch

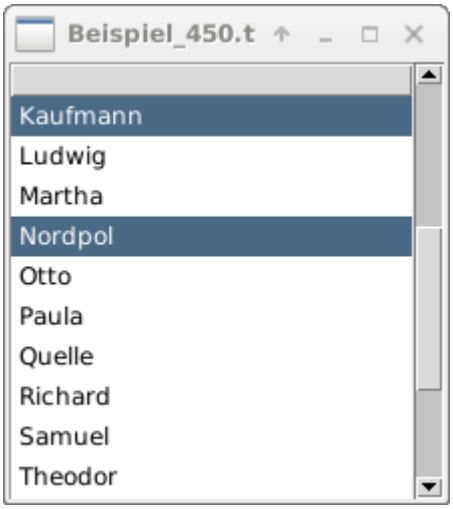
kann jeder Eintrag später gezielt angesprochen werden. In Zeile 19 wird programmseitig der Eintrag mit der ID `15` selektiert (markiert). In Zeile 20 wird die Anzeige des Treeviews auf den selektierten Eintrag verschoben. Nach einer Pause von zwei Sekunden wird in Zeile 25 die Selektion aufgehoben. In den Zeilen 26 und 27 wird der Eintrag mit der ID `10` selektiert und angezeigt.

Listing 41.70: Mehrere Einträge programmseitig selektieren (Beispiel450.tcl)

```
1  #!/usr/bin/env wish
2
3  set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav ↩
      Heinrich Ida Julius Kaufmann}
4  set Liste [concat $Liste {Ludwig Martha Nordpol Otto Paula↩
       Quelle Richard Samuel Theodor}]
5  set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe ↩
      Ypsilon Zacharias}]
6
7  ttk::treeview .tv -selectmode extended -columns Person ↩
      -show headings -yscrollcommand {.sbY set}
8
9  set ID 0
10 foreach Element $Liste {
11    .tv insert {} end -id $ID -values $Element
12    incr ID
13 }
14 ttk::scrollbar .sbY -command {.tv yview}
15
16 pack .sbY -side right -fill y
17 pack .tv -side left
18
19 .tv selection set {10 12 13 15}
20 .tv see 10
21 update idletasks
22
23 after 2000
24
25 .tv selection remove {12 15}
26 update idletasks
```



Und nach zwei Sekunden:



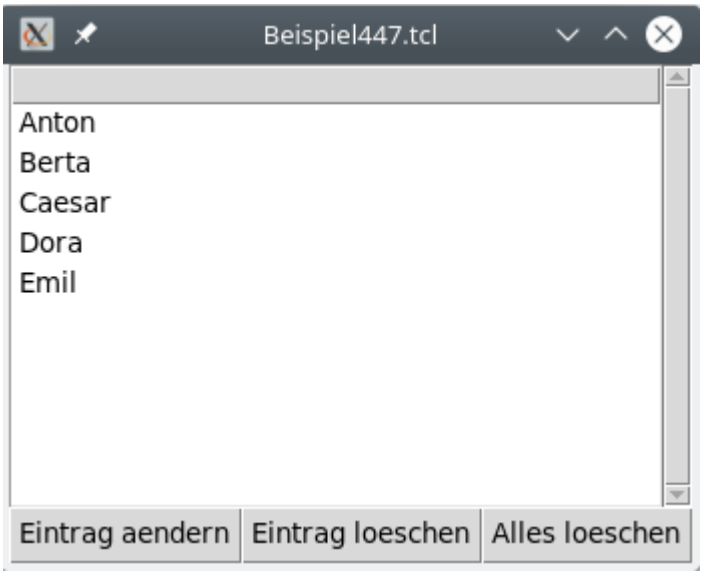
In Zeile 7 wird mit der Option `-selectmode extended` festgelegt, dass mehrere Einträge im Treeview selektiert werden können. In Zeile 19 werden vier Einträge selektiert. Die zu selektierenden Einträge werden als Liste übergeben. In Zeile 20 wird die Anzeige des Treeviews auf den ersten selektierten Eintrag verschoben. In Zeile 25 werden zwei Einträge aus der Selektion entfernt. Die zu entfernenden Einträge werden als Liste übergeben.

Listing 41.71: Einträge im Treeview ändern und löschen (Beispiel447.tcl)

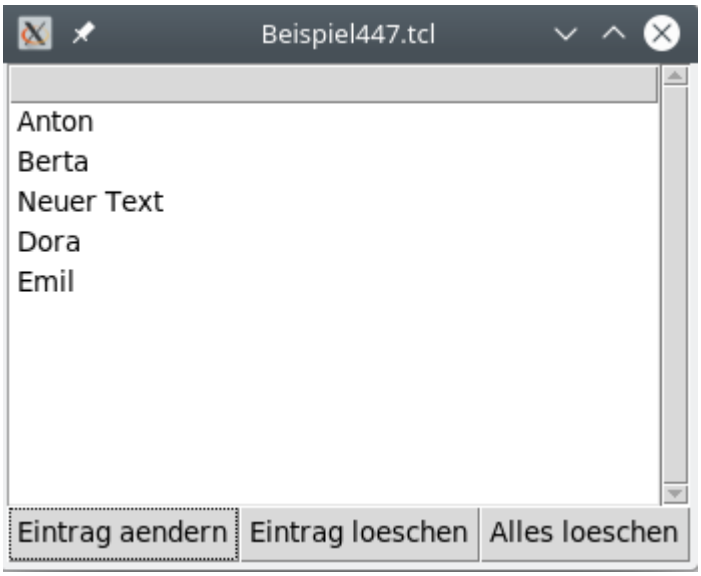
```
1  #!/usr/bin/env wish
2
3  proc EintragAendern {Liste Eintrag} {
4    set ID [lindex $Liste $Eintrag]
5    .frOben.tv item $ID -values {"Neuer Text"}
6    update idletasks
7  }
8
9  proc EintragLoeschen {Liste Eintrag} {
10   set ID [lindex $Liste $Eintrag]
```

```
11    .frOben.tv delete $ID
12    update idletasks
13
14    # ID-Liste anpassen
15    set AnzahlElemente [llength $Liste]
16    if {$Eintrag == 0} {
17      set Liste [lrange $Liste 1 end]
18    } elseif {$Eintrag == [expr $AnzahlElemente - 1]} {
19      set Liste [lrange $Liste 0 end-1]
20    } else {
21      set Liste [concat [lrange $Liste 0 [expr $Eintrag - ↩
          1]] [lrange $Liste [expr $Eintrag + 1] end]]
22    }
23    return $Liste
24  }
25
26  proc AllesLoeschen {} {
27          set Eintraege [.frOben.tv children {}]
28          foreach i $Eintraege {
29                  .frOben.tv delete $i
30          }
31          return {}
32  }
33
34  set Liste {Anton Berta Caesar Dora Emil}
35
36  set ListeTreeviewID {}
37  ttk::frame .frOben
38  ttk::treeview .frOben.tv -columns Person -show headings ↩
      -yscrollcommand {.frOben.sbY set}
39  foreach Element $Liste {
40    set ID [.frOben.tv insert {} end -values $Element]
41    lappend ListeTreeviewID $ID
42  }
43  ttk::scrollbar .frOben.sbY -command {.frOben.tv yview}
44
45  pack .frOben.sbY -side right -fill y
46  pack .frOben.tv -side left -expand yes -fill both
47
48  ttk::frame .frUnten
49  ttk::button .frUnten.btAendern -text "Eintrag aendern" ↩
      -command {EintragAendern $ListeTreeviewID 2}
50  ttk::button .frUnten.btLoeschen -text "Eintrag loeschen" ↩
      -command {set ListeTreeviewID [EintragLoeschen ↩
      $ListeTreeviewID 2]}
51  ttk::button .frUnten.btAllesLoeschen -text "Alles loeschen↩
      " -command {set ListeTreeviewID [AllesLoeschen]}
52  pack .frUnten.btAendern -side left
53  pack .frUnten.btLoeschen -side left
54  pack .frUnten.btAllesLoeschen -side left
55
56  pack .frOben -side top -expand yes -fill both
57  pack .frUnten -side bottom
```

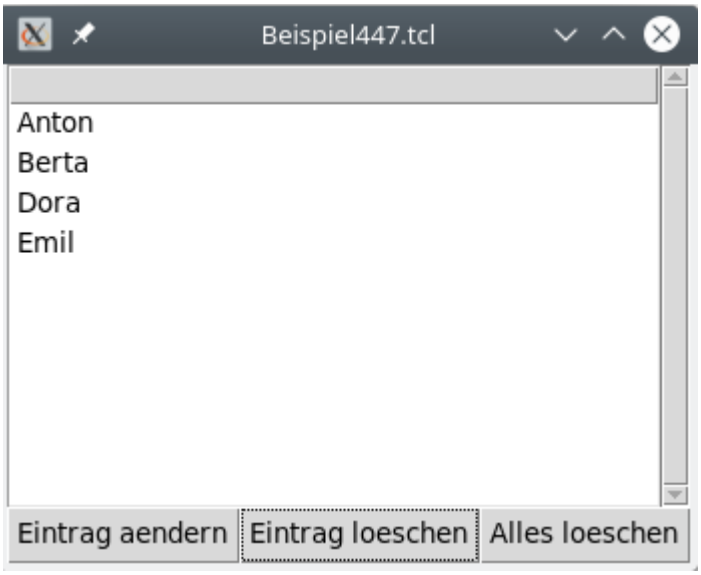
Nach dem Programmstart:



Nach einem Klick auf den Button `Eintrag ändern`:



Nach einem Klick auf den Button `Eintrag löschen`:



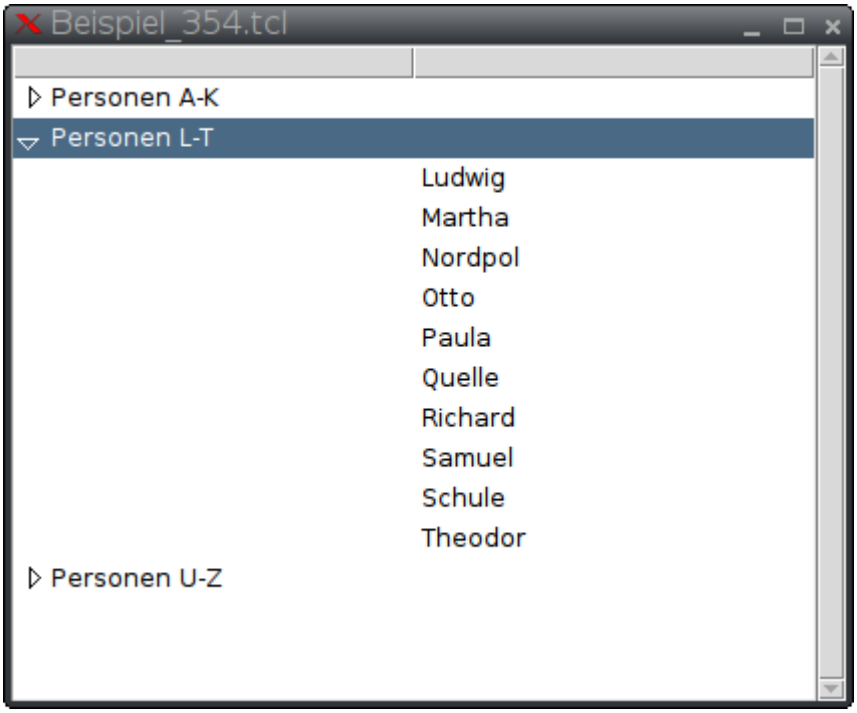
Nach einem Klick auf den Button `Alles löschen`:

In Zeile 36 wird eine leere Liste definiert, die die ID der Treeview-Einträge aufnehmen soll. In Zeile 40 wird beim Füllen des Treeview die (automatisch erzeugte) ID des Treeview-Eintrags gespeichert. In Zeile 41 wird diese ID der Liste mit allen Treeview-ID hinzugefügt. In Zeile 49 wird die Prozedur `EintragAendern` aufgerufen. An die Prozedur werden sowohl die Liste mit den ID der Treeview-Einträge übergeben als auch das zu ändernde Element. In Zeile 4 wird aus der Liste mit allen Treeview-ID die ID des zu ändernden Eintrags ermittelt. In Zeile 5 wird der Eintrag geändert. `$ID` enthält die ID des Treeview-Eintrags und `{"Neuer Text"}` enthält den neuen Eintrag. Dabei handelt es sich genau genommen um eine Liste, die die gesamte Zeile darstellt. In diesem Fall hat die Liste nur ein Element, weil der Treeview einspaltig ist. Wäre der Treeview mehrspaltig, könnte der Eintrag z. B. `{"Donald Duck" 1934 "Entenhausen"}` lauten. In Zeile 50 wird die Prozedur `EintragLoeschen` aufgerufen. An die Prozedur werden sowohl die Liste mit

den ID der Treeview-Einträge übergeben als auch das zu löschende Element. Die Prozedur gibt nach dem Löschen die geänderte Liste der ID mit den Treeview-Einträgen zurück (also ohne das gelöschte Element), die wieder in die Variable `ListeTreeviewID` gespeichert wird. In Zeile 10 wird aus der Liste mit allen Treeview-ID die ID des zu löschenden Eintrags ermittelt. In Zeile 11 wird der Eintrag gelöscht. In den Zeilen 15 bis 22 wird das Element in der Liste mit den Treeview-Einträgen gelöscht. Wenn es sich um das erste Element handelt, wird die Zeile 17 ausgeführt. Wenn es sich um das letzte Element handelt, wird Zeile 19 ausgeführt. Wenn es sich um ein anderes Element handelt, wird Zeile 21 ausgeführt. In Zeile 23 wird die neue Liste (also ohne das gelöschte Element) an das Hauptprogramm zurückgegeben. In Zeile 51 wird die Prozedur `AllesLoeschen` aufgerufen. In Zeile 27 werden alle Einträge des Treeviews als Liste gespeichert. In den Zeilen 28 bis 30 werden alle Einträge durchlaufen und gelöscht (Zeile 29). Zeile 31 gibt eine leere Liste zurück.

Listing 41.72: Treeview mit Knoten (Beispiel354.tcl)

```
#!/usr/bin/env wish

set Liste1 {Anton Berta Caesar Dora Emil Friedrich Gustav Heinrich Ida Julius Kaufmann}
set Liste2 {Ludwig Martha Nordpol Otto Paula Quelle Richard Samuel Schule Theodor}
set Liste3 {Ulrich Viktor Wilhelm Xanthippe Ypsilon Zacharias}

ttk::treeview .tv -columns Person -yscrollcommand {.sbY set}
.tv insert {} end -id Personen1 -text "Personen A-K"
.tv insert {} end -id Personen2 -text "Personen L-T"
.tv insert {} end -id Personen3 -text "Personen U-Z"

foreach Element $Liste1 {
  .tv insert Personen1 end -values $Element
}

foreach Element $Liste2 {
  .tv insert Personen2 end -values $Element
}

foreach Element $Liste3 {
  .tv insert Personen3 end -values $Element
}

ttk::scrollbar .sbY -command {.tv yview}

pack .sbY -side right -fill y
pack .tv -side left -expand yes -fill both
```

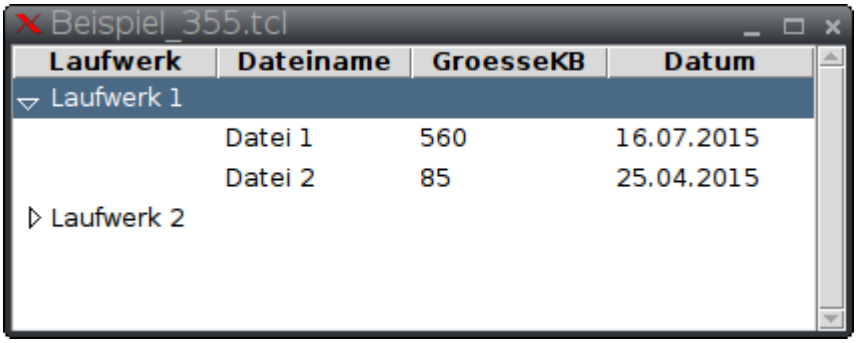


In den Zeilen 3 bis 5 werden drei Namenslisten definiert. In den Zeilen 8 bis 10 werden drei Einträge in der ersten Spalte festgelegt, die später als Knoten dienen. In der Zeile 13 werden die Personen aus der `Liste1` dem ersten Knoten `Personen1` untergeordnet. In der Zeile 27 werden die Optionen `-expand yes` und `-fill both` ergänzt, damit der Treeview beim Vergrößern des Dialogs ebenfalls größer wird.

Listing 41.73: Treeview mit Knoten und mit mehreren Spalten (Beispiel355.tcl)

```
1  #!/usr/bin/env wish
2
3  set Dateien1 {{"Datei 1" "560" "16.07.2015"} {"Datei 2" "
     85" "25.04.2015"}}
4  set Dateien2 {{"Datei 3" "89" "03.11.2014"} {"Datei 4" "
     793" "15.08.2015"} {"Datei 5" "688" "30.11.2014"}}
5
6  ttk::treeview .tv -columns {Dateiname GroesseKB Datum}
     -yscrollcommand {.sbY set}
7
8  .tv heading #0 -text "Laufwerk"
9  .tv heading Dateiname -text "Dateiname"
10 .tv heading GroesseKB -text "GroesseKB"
11 .tv heading Datum -text "Datum"
12
13 .tv insert {} end -id Laufwerk1 -text "Laufwerk 1"
14 .tv insert {} end -id Laufwerk2 -text "Laufwerk 2"
15
16 foreach Element $Dateien1 {
17   .tv insert Laufwerk1 end -values $Element
18 }
19
```

```
20 foreach Element $Dateien2 {
21   .tv insert Laufwerk2 end -values $Element
22 }
23
24 ttk::scrollbar .sbY -command {.tv yview}
25
26 pack .sbY -side right -fill y
27 pack .tv -side left -expand yes -fill both
```

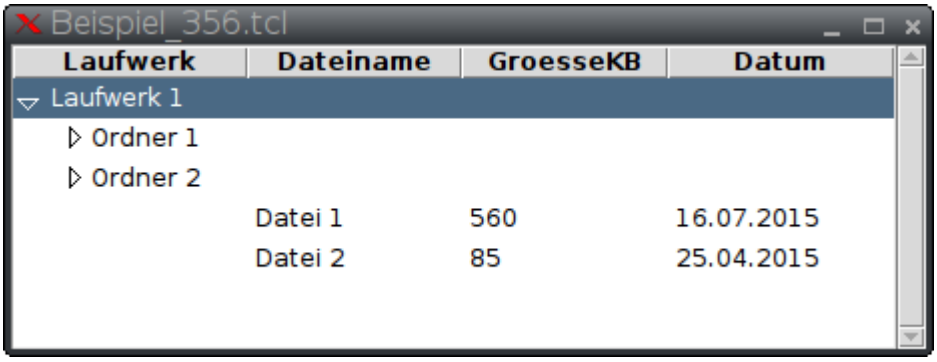


In den Zeilen 3 und 4 werden Dateilisten definiert, die Dateiname, Dateigröße und Dateidatum enthalten. In der Zeile 6 wird festgelegt, dass der Treeview drei Spalten haben soll. Die erste Spalte mit den Knoten wird hierbei nicht aufgeführt. Auch die Option `-show "headings"` wird nicht hinzugefügt, weil sonst die Spalte mit den Knoten nicht dargestellt wird. In Zeile 8 wird die Überschrift für die erste Spalten (Index 0) festgelegt. In den Zeilen 9 bis 11 werden die restlichen Überschriften festgelegt.

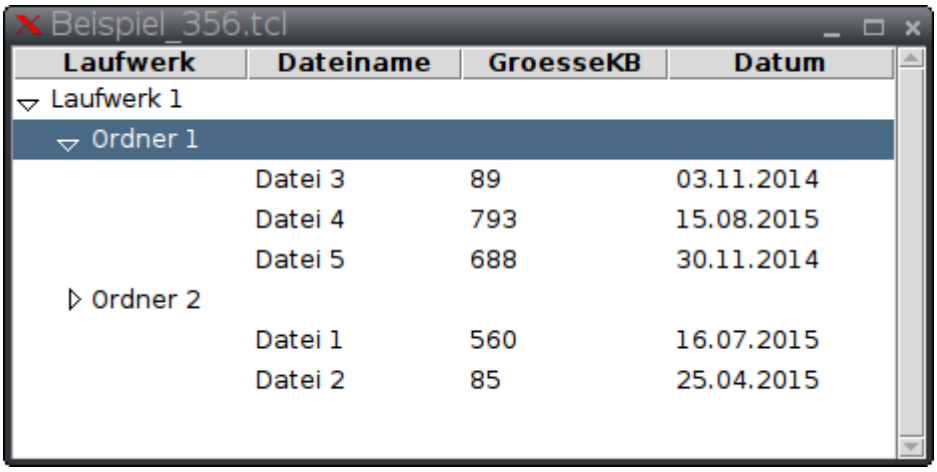
Listing 41.74: Treeview mit Knoten und Unterknoten (Beispiel356.tcl)

```
1  #!/usr/bin/env wish
2
3  set Dateien1 {{"Datei 1" "560" "16.07.2015"} {"Datei 2" "
     85" "25.04.2015"}}
4  set Dateien2 {{"Datei 3" "89" "03.11.2014"} {"Datei 4" "
     793" "15.08.2015"} {"Datei 5" "688" "30.11.2014"}}
5  set Dateien3 {{"Datei 6" "447" "17.09.2013"} {"Datei 7" "
     587" "02.03.2014"}}
6
7  ttk::treeview .tv -columns {Dateiname GroesseKB Datum}
     -yscrollcommand {.sbY set}
8
9  .tv heading #0 -text "Laufwerk"
10 .tv heading Dateiname -text "Dateiname"
11 .tv heading GroesseKB -text "GroesseKB"
12 .tv heading Datum -text "Datum"
13
14 .tv insert {} end -id Laufwerk1 -text "Laufwerk 1"
15 .tv insert Laufwerk1 end -id Ordner1 -text "Ordner 1"
16 .tv insert Laufwerk1 end -id Ordner2 -text "Ordner 2"
17
18 foreach Element $Dateien1 {
19   .tv insert Laufwerk1 end -values $Element
```

```
20 }
21
22 foreach Element $Dateien2 {
23   .tv insert Ordner1 end -values $Element
24 }
25
26 foreach Element $Dateien3 {
27   .tv insert Ordner2 end -values $Element
28 }
29
30 ttk::scrollbar .sbY -command {.tv yview}
31
32 pack .sbY -side right -fill y
33 pack .tv -side left -expand yes -fill both
```



Mit aufgeklapptem `Ordner 1`:



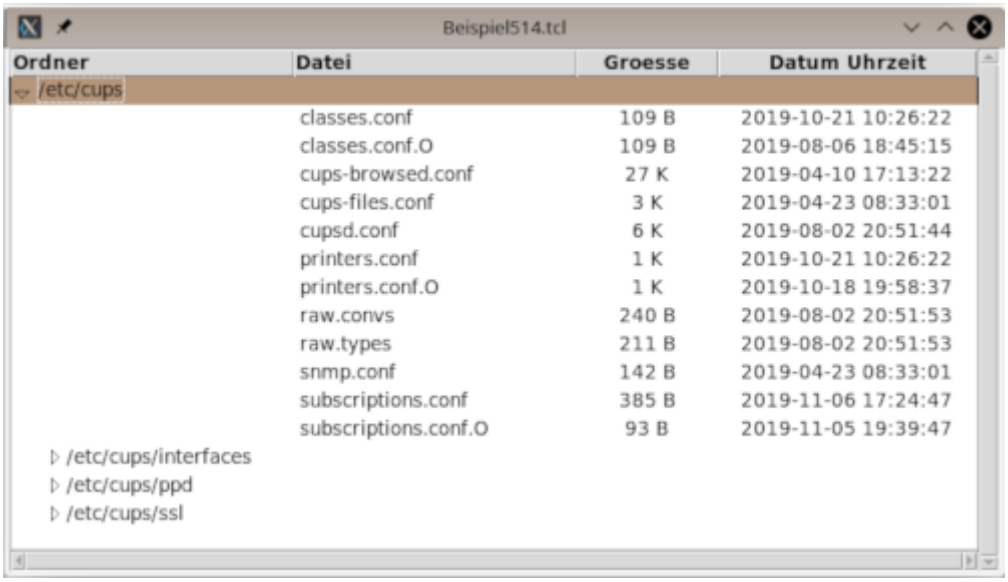
In Zeile 14 wird der Knoten `Laufwerk1` festgelegt. In den Zeilen 15 und 16 werden die Unterknoten `Ordner1` und `Ordner2` festgelegt. In den Zeilen 18 bis 20 werden Dateien dem (Haupt-)Knoten `Laufwerk1` zugeordnet. In den Zeilen 22 bis 24 werden Dateien dem (Unter-)Knoten `Ordner1` zugeordnet.
Beachten Sie folgende Besonderheit bei der Darstellung eines Knotens: das Symbol zum Auf- und Zuklappen wird nur dann angezeigt, wenn der Knoten Unterelemente enthält.

Ein Knoten ohne Unterelement hat kein Symbol. Das folgende Beispiel zeigt, wie man mit einem Dummy-Eintrag dafür sorgt, dass immer ein Symbol neben dem Knoten angezeigt wird.

Listing 41.75: Treeview mit nachladendem Inhalt und virtuellen Ereignissen (Beispiel514.tcl)

```
#!/usr/bin/env wish

proc GroesseFormatieren {Groesse} {
        if {$Groesse > 1048576} {
                set GroesseFormatiert "[expr round(1.0*$Groesse/1048576)] M"
        } elseif {$Groesse > 1024} {
                set GroesseFormatiert "[expr round(1.0*$Groesse/1024)] K"
        } else {
                set GroesseFormatiert "$Groesse B"
        }
        return $GroesseFormatiert
}

proc KnotenOeffnen {ID} {
        if {[.tv exists Dummy$ID]} {
                .tv delete Dummy$ID
        }
        set Startordner [.tv item $ID -text]
        set Dateien [lsort -nocase [glob -nocomplain -directory $Startordner -types {f} *]]
        foreach Element $Dateien {
                set Name [file tail $Element]
                set Groesse [GroesseFormatieren [file size $Element]]
                set Datum [clock format [file mtime $Element] -format {%Y-%m-%d %H:%M:%S}]
                set Eintrag {}
                lappend Eintrag $Name
                lappend Eintrag $Groesse
                lappend Eintrag $Datum
                set tmpID [.tv insert $ID end -values $Eintrag]
        }
        set Ordner [lsort -nocase [glob -nocomplain -directory $Startordner -types {d} *]]
        foreach Element $Ordner {
                set tmpID [.tv insert $ID end -text $Element]
                .tv insert $tmpID end -id Dummy$tmpID -text ""
        }
}

proc KnotenSchliessen {ID} {
        set Elemente [.tv children $ID]
```

```
39          foreach Element $Elemente {
40                  if {[.tv exists $Element]} {
41                          .tv delete $Element
42                  }
43          }
44          .tv insert $ID end -id Dummy$ID -text ""
45 }
46
47 set Startordner $env(HOME)
48
49 ttk::treeview .tv -columns {Datei Groesse Datum} ↲
      -yscrollcommand {.sbY set} -xscrollcommand {.sbX set}
50 .tv column #0 -minwidth 200 -stretch 1
51 .tv column Datei -minwidth 200 -stretch 1
52 .tv column Groesse -minwidth 100 -stretch 1
53 .tv column Datum -minwidth 150 -stretch 1
54
55 .tv heading #0 -text "Ordner"
56 .tv heading Datei -text "Datei"
57 .tv heading Groesse -text "Groesse"
58 .tv heading Datum -text "Datum Uhrzeit"
59
60 .tv heading #0 -anchor w
61 .tv heading Datei -anchor w
62 .tv heading Groesse -anchor c
63 .tv heading Datum -anchor c
64
65 .tv column #0 -anchor w
66 .tv column Datei -anchor w
67 .tv column Groesse -anchor c
68 .tv column Datum -anchor c
69
70 set ID [.tv insert {} end -text $Startordner]
71 .tv insert $ID end -id Dummy$ID -text ""
72
73 ttk::scrollbar .sbY -command {.tv yview}
74 ttk::scrollbar .sbX -orient horizontal -command {.tv xview↲
      }
75
76 pack .sbY -side right -fill y
77 pack .sbX -side bottom -fill x
78 pack .tv -side left -expand yes -fill both
79 bind .tv <<TreeviewOpen>> {KnotenOeffnen [.tv selection]}
80 bind .tv <<TreeviewClose>> {KnotenSchliessen [.tv ↲
      selection]}
```

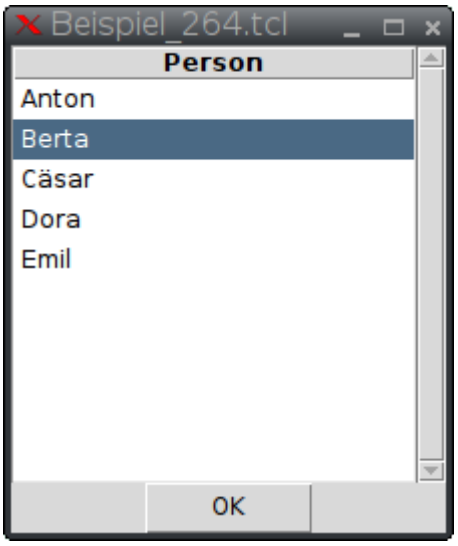


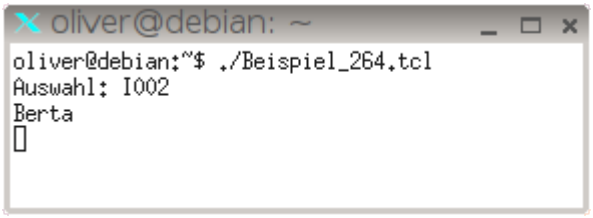
Es kann sinnvoll sein, beim Erstellen des Treeviews nicht sofort den gesamten Inhalten (inklusive des Inhalts aller Unterknoten) zu ermitteln. Stattdessen kann man den Inhalt eines Knotens erst dann ermitteln, wenn der Knoten geöffnet wird. Dazu braucht man die beiden virtuellen Ereignisse `<<TreeviewOpen>>` und `<<TreeviewClose>>`.
In den Zeilen 47 bis 80 wird ein Treeview erstellt, der als einzigen Ordner nur den zugeklappten Startordner enthält (Zeile 70). Damit der Treeview neben dem Knoten das Symbol zum Auf- und Zuklappen anzeigt, muss der Knoten ein Unterelement haben. Dazu wird in Zeile 71 ein Dummy-Eintrag erzeugt. In den Zeilen 79 und 80 werden die beiden virtuellen Ereignisse `<<TreeviewOpen>>` und `<<TreeviewClose>>` definiert.
Wenn der Benutzer einen Knoten öffnet, wird die Prozedur `KnotenOeffnen` (Zeilen 14 bis 35) aufgerufen. In den Zeilen 15 bis 17 wird zuerst der Dummy-Eintrag gelöscht. In Zeile 18 wird der Startordner aus dem vom Benutzer angeklickten Knoten ermittelt. In den Zeilen 19 bis 29 werden alle Dateien in dem Ordner mit den Dateieigenschaften bestimmt und in den Treeview eingetragen. In den Zeilen 30 bis 34 werden alle Unterordner ermittelt und hinzugefügt.
Wenn der Benutzer einen Knoten schließt, wird die Prozedur `KnotenSchliessen` (Zeilen 37 bis 45) aufgerufen. In den Zeilen 38 bis 43 werden alle Unterelemente des Knotens gelöscht. In Zeile 44 wird wieder ein Dummy-Eintrag dem Knoten hinzugefügt, damit das Symbol zum Auf- und Zuklappen angezeigt wird.

Listing 41.76: Treeview als Ersatz für eine Listbox: es kann genau ein Element ausgewählt werden (Beispiel264.tcl)

```
1 #!/usr/bin/env wish
2
3 set Liste {Anton Berta Caesar Dora Emil}
4
5 proc AuswahlAnzeigen {} {
6   set Auswahl [.fr.tv selection]
7   puts "Auswahl: $Auswahl"
8   foreach Element $Auswahl {
9     puts [.fr.tv item $Element -value]
```

```
10    }
11 }
12
13 ttk::frame .fr
14 ttk::treeview .fr.tv -columns VName -show headings ↲
     -selectmode browse -yscrollcommand {.fr.sbY set}
15 foreach Element $Liste {
16    .fr.tv insert {} end -values $Element
17 }
18 .fr.tv heading VName -text "Person"
19 ttk::scrollbar .fr.sbY -command {.fr.tv yview}
20 ttk::button .bt -text "OK" -command AuswahlAnzeigen
21
22 pack .fr.sbY -side right -fill y
23 pack .fr.tv -side left
24 pack .fr
25 pack .bt
```



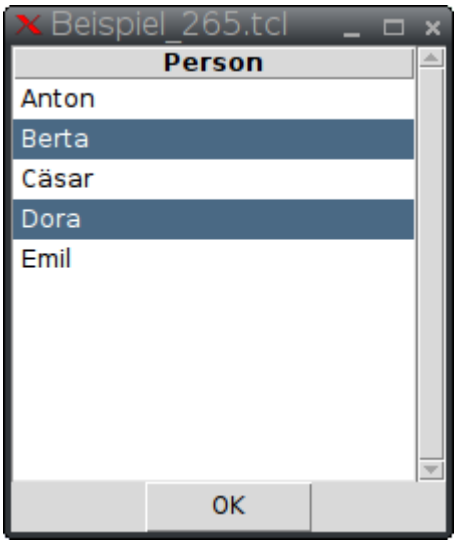


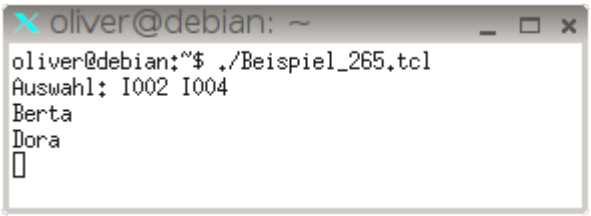
In Zeile 14 wird durch die Option `-selectmode browse` festgelegt, dass genau ein Element ausgewählt werden kann. In Zeile 7 werden die ID der selektierten Elemente angezeigt. In Zeile 9 werden die selektierten Elemente angezeigt.

Listing 41.77: Treeview als Ersatz für eine Listbox: es können mehrere Elemente ausgewählt werden (Beispiel265.tcl)

```
1 #!/usr/bin/env wish
```

```
2
3  set Liste {Anton Berta Caesar Dora Emil}
4
5  proc AuswahlAnzeigen {} {
6    set Auswahl [.fr.tv selection]
7    puts "Auswahl: $Auswahl"
8    foreach Element $Auswahl {
9      puts [.fr.tv item $Element -value]
10   }
11 }
12
13 ttk::frame .fr
14 ttk::treeview .fr.tv -columns VName -show headings ↲
     -selectmode extended -yscrollcommand {.fr.sbY set}
15 foreach Element $Liste {
16   .fr.tv insert {} end -values $Element
17 }
18 .fr.tv heading VName -text "Person"
19 ttk::scrollbar .fr.sbY -command {.fr.tv yview}
20 ttk::button .bt -text "OK" -command AuswahlAnzeigen
21
22 pack .fr.sbY -side right -fill y
23 pack .fr.tv -side left
24 pack .fr
25 pack .bt
```

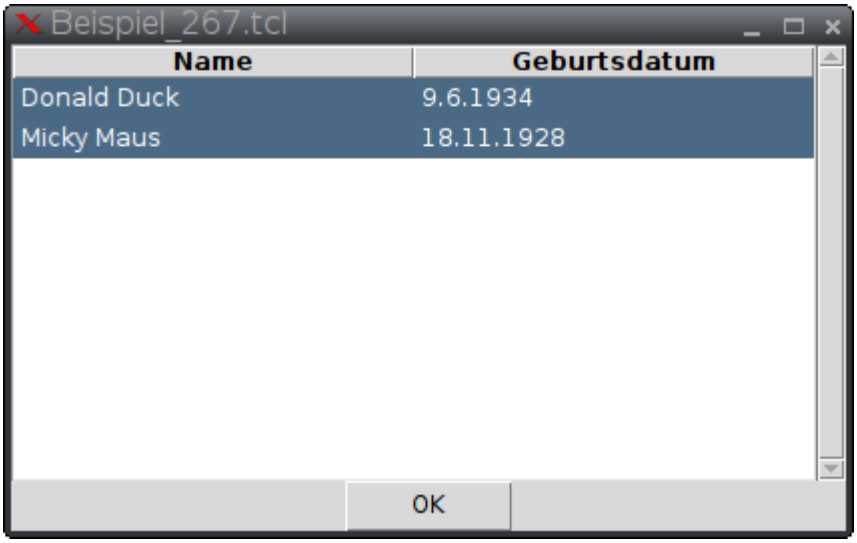


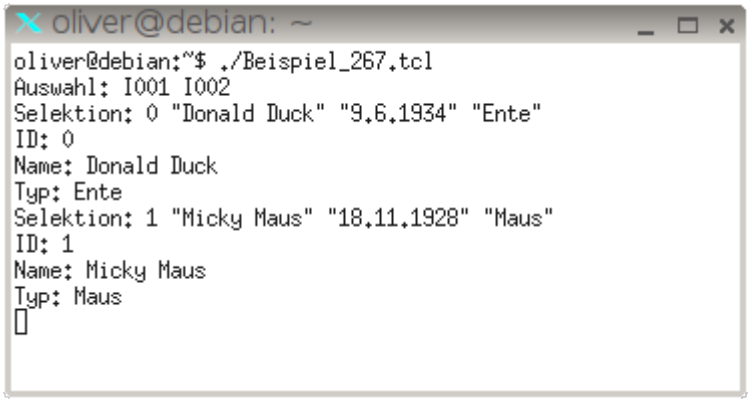


In Zeile 14 wird durch die Option `-selectmode extended` festgelegt, dass (mit der `Strg`-Taste) mehrere Elemente ausgewählt werden können. In Zeile 7 werden die ID der selektierten Elemente angezeigt. In Zeile 9 werden die selektierten Elemente angezeigt.

Listing 41.78: Treeview als Ersatz für eine Listbox mit mehreren Spalten und Index-ID für den Zugriff auf die Liste (Beispiel267.tcl)

```
1  #!/usr/bin/env wish
2
3  set Liste {{0 "Donald Duck" "9.6.1934" "Ente"} {1 "Micky ↩
     Maus" "18.11.1928" "Maus"}}
4
5  proc AuswahlAnzeigen {Liste} {
6    set Auswahl [.fr.tv selection]
7    puts "Auswahl: $Auswahl"
8    foreach Element $Auswahl {
9      set Wert [.fr.tv item $Element -value]
10     puts "Selektion: $Wert"
11     set ID [lindex $Wert 0]
12
13         set Name [lindex $Liste $ID 1]
14         set Typ [lindex $Liste $ID 3]
15
16     puts "ID: $ID"
17     puts "Name: $Name"
18     puts "Typ: $Typ"
19   }
20 }
21
22 ttk::frame .fr
23 ttk::treeview .fr.tv -columns {ID Name Datum} ↩
     -displaycolumns {Name Datum} -show headings -selectmode↩
      extended -yscrollcommand {.fr.sbY set}
24 foreach Element $Liste {
25   .fr.tv insert {} end -values $Element
26 }
27 .fr.tv heading Name -text "Name"
28 .fr.tv heading Datum -text "Geburtsdatum"
29 ttk::scrollbar .fr.sbY -command {.fr.tv yview}
30 ttk::button .bt -text "OK" -command {AuswahlAnzeigen ↩
     $Liste}
31
32 pack .fr.sbY -side right -fill y
33 pack .fr.tv -side left
34 pack .fr
35 pack .bt
```





Üblicherweise basiert der Treeview auf einer Liste mit Einträgen. Es ist deshalb praktisch, wenn man den Index der Liste auch im Treeview zur Verfügung hat, so dass man später über diesen Index auf die Liste zugreifen kann. In Zeile 3 erhalten die Datensätze eine ID, Name, Geburtsdatum und Typ. In Zeile 21 werden die Spalten des Treeview definiert. Mit der Option `-displaycolumns {Name Datum}` wird bestimmt, dass nur die Spalten `Name` und `Datum` angezeigt werden. Die weiteren Spalten der Liste (in diesem Fall der `Typ`) werden nicht im Treeview verwendet. In Zeile 30 wird beim einem Klick auf den `OK`-Button die Liste an die Prozedur übergeben. In Zeile 11 wird die ID des selektierten Treeview-Eintrags ermittelt. Anhand dieser ID wird in den Zeilen 13 und 14 auf die Liste zugegriffen. In Zeile 18 wird der Typ gemäß der Liste ausgegeben, ohne dass der Typ zuvor dem Treeview übergeben werden musste.

41.17.1 Treeview mit Tree kombinieren

Im Kapitel 17 wurde zum Speichern hierarchischer Strukturen der `Tree` vorgestellt. Das folgende Beispiel zeigt, wie man den Inhalt des `Trees` in einem `Treeview` darstellt.

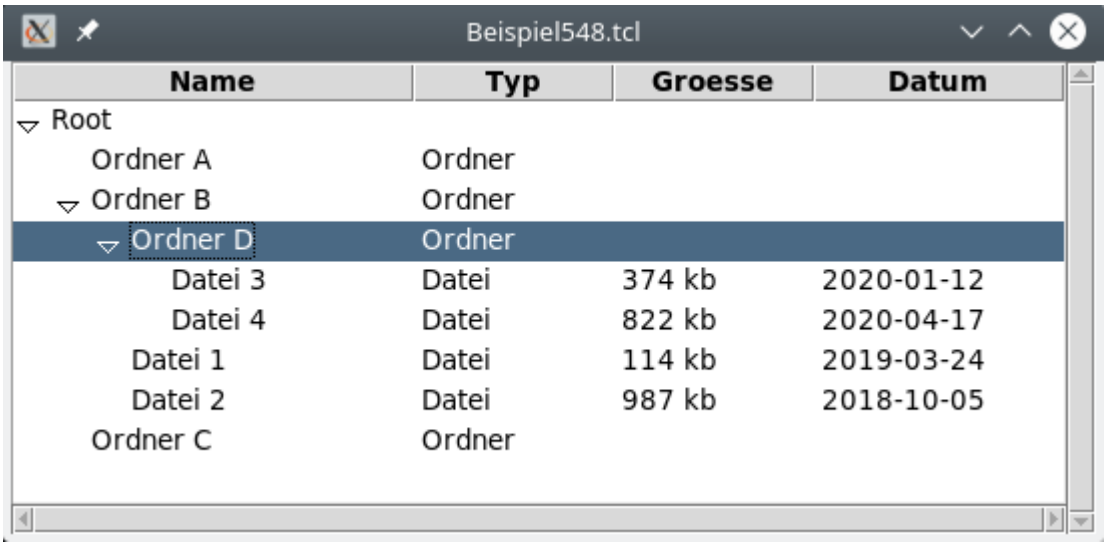
Listing 41.79: Tree als Treeview anzeigen (Beispiel548.tcl)

```
1 #!/usr/bin/env wish
```

```
2
3 package require struct::tree
4
5 ::struct::tree Baum
6
7 # Es wird folgender Baum erfasst:
8 # root -+- Ordner A
9 #        +- Ordner B -+- Ordner D -+- Datei 3
10 #        |            |             +- Datei 4
11 #        |            +- Datei 1
12 #        |            +- Datei 2
13 #        +- Ordner C
14
15 # Unterhalb von root:
16 Baum insert root end 0
17 Baum set 0 Name "Ordner A"
18 Baum set 0 Typ "Ordner"
19 Baum set 0 Groesse ""
20 Baum set 0 Datum ""
21
22 Baum insert root end 1
23 Baum set 1 Name "Ordner B"
24 Baum set 1 Typ "Ordner"
25 Baum set 1 Groesse ""
26 Baum set 1 Datum ""
27
28 Baum insert root end 2
29 Baum set 2 Name "Ordner C"
30 Baum set 2 Typ "Ordner"
31 Baum set 2 Groesse ""
32 Baum set 2 Datum ""
33
34 # Unterhalb von Ordner B:
35 Baum insert 1 end 3
36 Baum set 3 Name "Ordner D"
37 Baum set 3 Typ "Ordner"
38 Baum set 3 Groesse ""
39 Baum set 3 Datum ""
40
41 Baum insert 1 end 4
42 Baum set 4 Name "Datei 1"
43 Baum set 4 Typ "Datei"
44 Baum set 4 Groesse "114 kb"
45 Baum set 4 Datum "2019-03-24"
46
47 Baum insert 1 end 5
48 Baum set 5 Name "Datei 2"
49 Baum set 5 Typ "Datei"
50 Baum set 5 Groesse "987 kb"
51 Baum set 5 Datum "2018-10-05"
52
53 # Unterhalb von Ordner D:
54 Baum insert 3 end 6
```

```
55 Baum set 6 Name "Datei 3"
56 Baum set 6 Typ "Datei"
57 Baum set 6 Groesse "374 kb"
58 Baum set 6 Datum "2020-01-12"
59
60 Baum insert 3 end 7
61 Baum set 7 Name "Datei 4"
62 Baum set 7 Typ "Datei"
63 Baum set 7 Groesse "822 kb"
64 Baum set 7 Datum "2020-04-17"
65
66 #################
67
68 ttk::treeview .tv -columns {Typ Groesse Datum} ↩
      -yscrollcommand {.sbY set} -xscrollcommand {.sbX set}
69
70 .tv column #0 -minwidth 200 -stretch 1
71 .tv column Typ -minwidth 100 -stretch 1
72 .tv column Groesse -minwidth 100 -stretch 1
73 .tv column Datum -minwidth 100 -stretch 1
74
75 .tv heading #0 -text "Name"
76 .tv heading Typ -text "Typ"
77 .tv heading Groesse -text "Groesse"
78 .tv heading Datum -text "Datum"
79
80 .tv insert {} end -id root -text "Root" ; # die ID muss "↩
      root" lauten, weil in struct::tree der erste Knoten die↩
       ID "root" hat.
81
82 ttk::scrollbar .sbY -command {.tv yview}
83 ttk::scrollbar .sbX -orient horizontal -command {.tv xview↩
      }
84
85 pack .sbY -side right -fill y
86 pack .sbX -side bottom -fill x
87 pack .tv -side left -expand yes -fill both
88
89 #################
90
91 Baum walk root -order pre -type bfs Id {
92   set Name ""
93   set Typ ""
94   set Groesse ""
95   set Datum ""
96
97   if {[Baum keyexists $Id Name]} {
98     set Name [Baum get $Id Name]
99   }
100   if {[Baum keyexists $Id Typ]} {
101     set Typ [Baum get $Id Typ]
102   }
103   if {[Baum keyexists $Id Groesse]} {
```

```
104      set Groesse [Baum get $Id Groesse]
105    }
106    if {[Baum keyexists $Id Datum]} {
107      set Datum [Baum get $Id Datum]
108    }
109
110    set ElternId [Baum parent $Id]
111    if {$Typ eq "Ordner" || $Typ eq "Datei"} {
112      set Liste [list $Typ $Groesse $Datum]
113      .tv insert $ElternId end -id $Id -text $Name -values ↩
         $Liste
114    }
115  }
```



In den Zeilen 3 bis 64 wird der `Tree` definiert. In den Zeilen 68 bis 87 wird der `Treeview` erstellt. In den Zeilen 91 bis 115 wird der Inhalt des `Trees` in den `Treeview` eingefügt.

41.18 Text

Wenn man einen (längeren) Text anzeigen oder eingeben möchte, verwendet man das Text-Element, das sehr viele Funktionen ähnlich einer kleinen Textverarbeitung bietet (z. B. Textabschnitte unterschiedlich formatieren oder Bilder einbetten). Im Folgenden werden nur die wichtigsten Eigenschaften dargestellt und darüber hinaus auf die offizielle Dokumentation verwiesen. Das Text-Element gibt es nur als klassisches Element, nicht als ttk-Element.
Das Text-Element stellt den Text von links oben nach rechts unten dar. Die erste Zeile hat den Index 1 (nicht 0), das erste Zeichen innerhalb einer Zeile hat den Index 0.

Tabelle 41.27: Die wichtigsten Optionen

Option	Beschreibung
`-height Zeilen`	Anzahl der Zeilen
`-width Zeichen`	Anzahl der Zeichen pro Zeile

`-wrap none`
`-wrap char`
`-wrap word`
	Zeilenumbruch
	`none` = kein Zeilenumbruch
	`char` = nach einem Zeichen
	`word` = nur ganzes Wort

`-font Schrift`	Schrift
`-foreground Farbe`	Textfarbe
`-background Farbe`	Hintergrundfarbe
`-tabs "Position Art"`	Tabulator-Stopp definieren. Position (z. B.): `3c` = 3 Zentimeter `2i` = 2 Inch Art: `left` = linksbündig `right` = rechtsbündig `center` = zentriert `numeric` = Dezimaltrennzeichen ist zentriert
`-tabs {Position1 Art1 Position2 Art2}`	Tabulator-Stopps

Tabelle 41.27: Die wichtigsten Optionen

Option	Beschreibung
`.tx insert Zeile.Stelle Text`	Text einfügen (funktioniert nur im Status normal). Man darf als Position anstelle `Zeile.Stelle` auch folgende Schlüsselwörter verwenden (beachten Sie die Anführungszeichen). `insert` = aktuelle Cursorposition `"insert linestart"` = Zeilenanfang `"insert lineend"` = Zeilenende `end` = Textende Außerdem kann man eine Anzahl an Zeichen zu der jeweiligen Position hinzuzählen oder abziehen (z. B.): `"insert linestart+2c"` = 2 Zeichen nach dem Zeilenanfang `"insert lineend-2c"` = 2 Zeichen vor dem Zeilenende
`.tx delete VonZeile.Stelle BisZeile.Stelle`	Text löschen
`.tx delete 1.0 2.0`	Erste Zeile löschen
`.tx delete 1.0 end`	Gesamten Text löschen
`[.tx count -lines 1.0 end]`	Anzahl der Zeilen
`[.tx count -chars Zeile.0 Zeile.end]`	Anzahl der Zeichen in einer Zeile
`[.tx count -chars 1.0 end]`	Gesamte Anzahl der Zeichen im Text-Element. Das Zeilenende-Zeichen wird mitgezählt.
`set Variable [.tx get 1.0 end]`	Inhalt des Text-Elements in eine Variable speichern
`[.tx index insert]`	Cursor-Position im Text im Format `Zeile.Spalte`. Das Wort `insert` ist ein dafür reserviertes Wort.

Tabelle 41.27: Die wichtigsten Optionen

Option	Beschreibung
`[.tx tag ranges sel]`	Anfang und Ende des selektierten Bereichs im Text im Format `Anfangszeile.Anfangsspalte Endezeile.Endespalte`. Das Wort `sel` ist ein dafür reserviertes Wort.

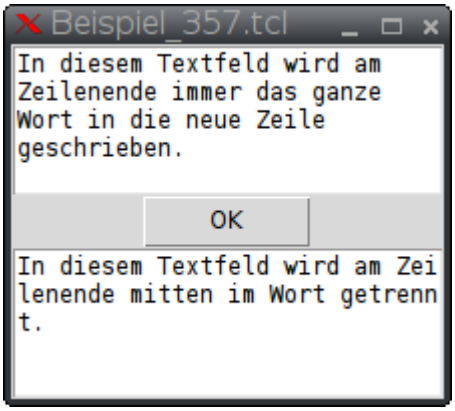
Mit folgenden Taste kann man im Text navigieren:

Tabelle 41.28: Tastensteuerung

Taste	Beschreibung
`Cursor`	links, rechts, hoch, runter
`Bild-hoch- / Bild-runter`	hoch, runter
`Pos1`	springt zum Zeilenanfang
`Ende`	springt zum Zeilenende
`Strg+Pos1`	springt zum Textanfang
`Strg+Ende`	springt zum Textende
`Shift+Cursor`	Text markieren
`Strg+c`	Text kopieren
`Strg+x`	Text ausschneiden
`Strg+v`	Text einfügen
`Tab`	Tabulator

Listing 41.80: Text eingeben und im Text navigieren (Beispiel357.tcl)

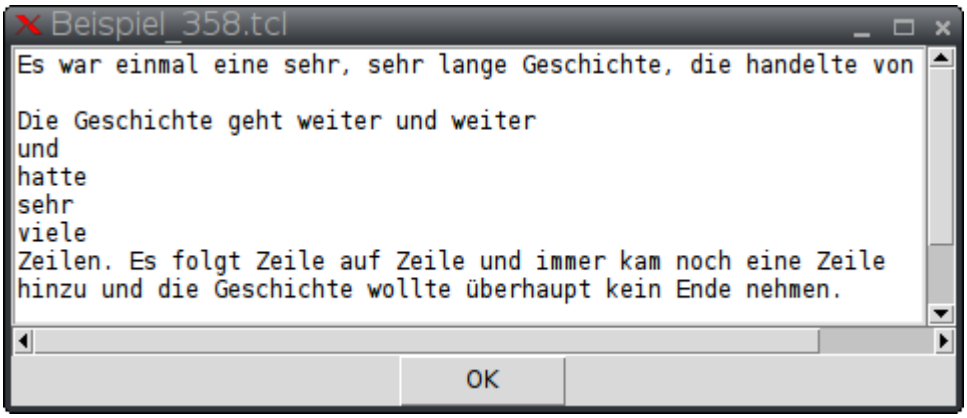
```
#!/usr/bin/env wish

text .txOben -height 5 -width 30 -wrap word
ttk::button .btOK -text "OK" -command exit
text .txUnten -height 5 -width 30 -wrap char

pack .txOben
pack .btOK
pack .txUnten
```



In Zeile 3 wird ein Textfeld definiert. Die Option `-wrap word` legt fest, dass beim Zeilenumbruch immer das ganze Wort in die neue Zeile gesetzt wird. In Zeile 5 wird ein zweites Textfeld definiert, dessen Zeilenumbruch bei jedem Zeichen erfolgen darf.

Listing 41.81: Text-Element mit Scroll-Leisten (Beispiel358.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::frame .fr
4  text .fr.tx -height 5 -width 15 -wrap none -xscrollcommand
       {.fr.sbX set} -yscrollcommand {.fr.sbY set}
5  ttk::scrollbar .fr.sbX -orient horizontal -command {.fr.tx
       xview}
6  ttk::scrollbar .fr.sbY -command {.fr.tx yview}
7
8  ttk::button .btOK -text "OK" -command exit
9
10 pack .fr -expand yes -fill both
11 pack .fr.sbX -side bottom -fill x
12 pack .fr.sbY -side right -fill y
13 pack .fr.tx -side top -expand yes -fill both
14
15 pack .btOK
```



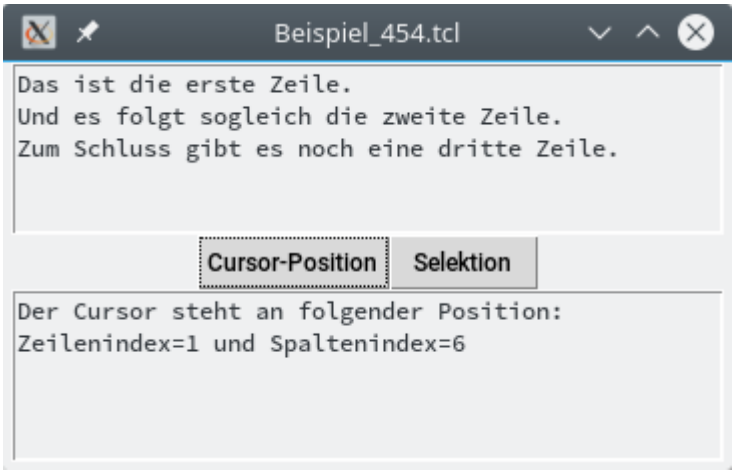
In Zeile 4 wird das Text-Element definiert. Damit man einen Effekt bei der waagrechten

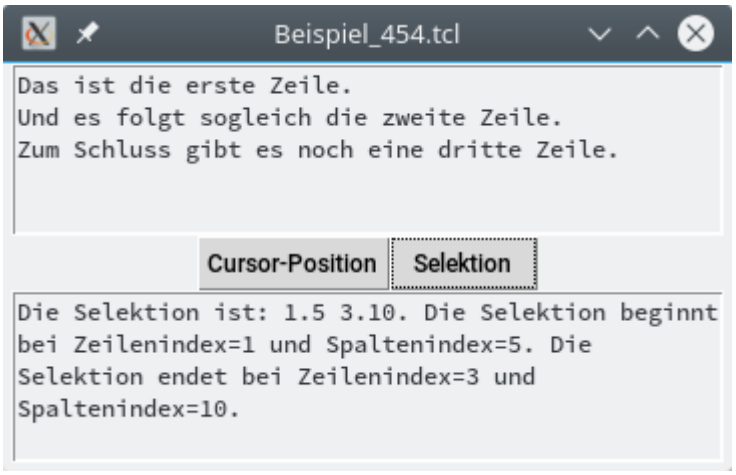
Scroll-Leiste sieht, wurde die Option `-wrap none` gewählt. Außerdem wurde das Text-Element zusammen mit den Scroll-Leisten in einen Frame gesetzt, weil die drei Elemente eine Einheit bilden. In Zeile 10 wird festgelegt, dass der Frame die Größe ändert, wenn man das Fenster vergrößert oder verkleinert. In Zeile 13 wird festgelegt, das auch das Text-Element die Größe ändert.

Listing 41.82: Cursor-Position und Selektion im Text (Beispiel454.tcl)

```
1  #!/usr/bin/env wish
2
3  proc CursorZeigen {} {
4          set CursorPosition [.txOben index insert]
5          set tmp [split $CursorPosition "."]
6          lassign $tmp CursorZeile CursorSpalte
7          .txUnten delete 1.0 end
8          .txUnten insert end "Der Cursor steht an folgender Position: Zeilenindex=$CursorZeile und Spaltenindex=$CursorSpalte\n"
9  }
10
11 proc SelektionZeigen {} {
12         set Selektion [.txOben tag ranges sel]
13         .txUnten delete 1.0 end
14         .txUnten insert end   "Die Selektion ist: $Selektion. "
15
16         set tmp [split $Selektion " "]
17         lassign $tmp Anfang Ende
18
19         set tmp [split $Anfang "."]
20         lassign $tmp AnfangZeile AnfangSpalte
21
22         set tmp [split $Ende "."]
23         lassign $tmp EndeZeile EndeSpalte
24
25         .txUnten insert end  "Die Selektion beginnt bei Zeilenindex=$AnfangZeile und Spaltenindex=$AnfangSpalte. "
26         .txUnten insert end  "Die Selektion endet bei Zeilenindex=$EndeZeile und Spaltenindex=$EndeSpalte."
27 }
28
29 text .txOben -height 5 -width 50 -wrap none
30
31 ttk::frame .fr
32 ttk::button .fr.btCursor -text "Cursor-Position" -command {CursorZeigen}
33 ttk::button .fr.btSelektion -text "Selektion" -command {SelektionZeigen}
34
35 text .txUnten -height 5 -width 50 -wrap word
36
```

```
37 pack .txOben -side top
38 pack .fr.btCursor -side left
39 pack .fr.btSelektion -side left
40 pack .fr -side top
41 pack .txUnten -side top
42
43 .txOben insert end "Das ist die erste Zeile.\n"
44 .txOben insert end "Und es folgt sogleich die zweite ↩
   Zeile.\n"
45 .txOben insert end "Zum Schluss gibt es noch eine dritte ↩
   Zeile."
```



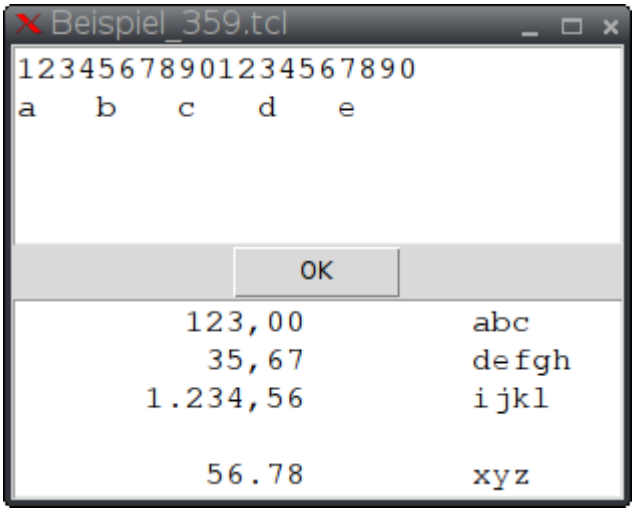


Die Zeilen 3 bis 9 umfassen die Prozedur `CursorZeigen`. In Zeile 4 wird die Position des Eingabe-Cursors abgefragt. Das Schlüsselwort `insert` repräsentiert die Eingabeposition. Das Ergebnis ist im Format `Zeile.Spalte`, wobei die erste Zeile den Wert `1` hat, die erste Spalte den Wert `0`. In Zeile 5 werden anhand des Punkts Zeile und Spalte voneinander getrennt. In Zeile 6 werden die Zeile und Spalte in zwei Variablen gespeichert. In Zeile 7 wird der Inhalt der unteren Text-Box gelöscht. In Zeile 8 erfolgt die Textausgabe in der unteren Text-Box.

Die Zeilen 11 bis 27 beinhalten die Prozedur `SelektionZeigen`. In Zeile 12 wird der selektierte Bereich ermittelt. Das Schlüsselwort `sel` repräsentiert dabei den selektierten Bereich. Die Rückgabe erfolgt in der Form
`Anfangszeile.Anfangsspalte Endezeile.Endespalte`.
In Zeile 13 wird der Inhalt der unteren Text-Box gelöscht. In Zeile 14 erfolgt die Textausgabe in der unteren Text-Box. In Zeile 16 werden die Positionsangaben der Selektion anhand des Leerzeichens in Anfangs- und Endeposition getrennt. In Zeile 17 werden die beiden Positionen in zwei Variablen gespeichert. In Zeile 19 wird die Anfangsposition anhand des Punkts in Zeile und Spalte aufgeteilt. In Zeile 20 werden Zeile und Spalte in zwei Variablen gespeichert. Die Zeilen 22 und 23 sind analog für die Ende-Position. In den Zeilen 25 und 26 erfolgt die Textausgabe in der unteren Text-Box.

Listing 41.83: Tabulator-Stopps festlegen (Beispiel359.tcl)

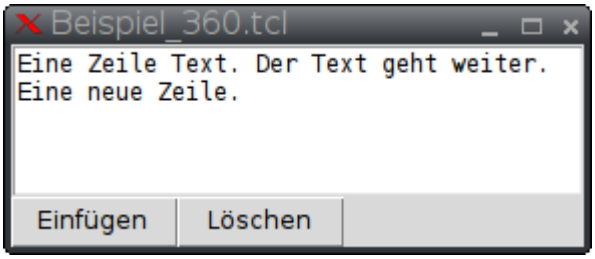
```
1 #!/usr/bin/env wish
2
3 text .txOben -height 5 -width 30 -font "courier 12" -tabs ↩
    "[expr {4 * [font measure {courier 12} 0]}] left" ↩
    -tabstyle wordprocessor
4 ttk::button .btOK -text "OK" -command exit
5 text .txUnten -height 5 -width 30 -font "courier 12" -tabs↩
     {3c numeric 6c left}
6 pack .txOben
7 pack .btOK
8 pack .txUnten
```



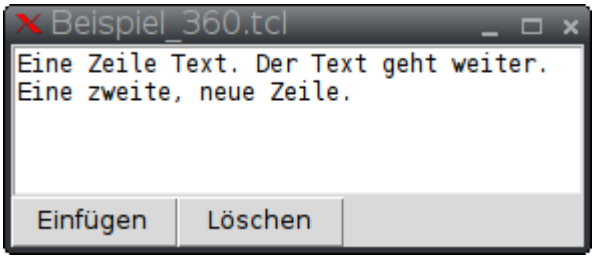
In Zeile 3 wird dem Text-Element mit der Option `-font "courier 12"` eine proportionale Schriftart zugewiesen. Die Option
`-tabs "[expr {4 ... }] left" -tabstyle wordprocessor`
legt fest, dass nach jeweils vier Zeichen ein Tabulator-Stopp folgt. In Zeile 5 wird ein weiteres Text-Element definiert. Es hat zwei Tabulator-Stopps an den Positionen `3c` (3 Zentimeter) und `6c` (6 Zentimeter). Der erste Tabulator-Stopp ist numerisch, d. h. die Zahl wird zentriert um das Dezimaltrennzeichen angeordnet. Als Dezimaltrennzeichen wird sowohl der Punkt als auch das Komma genommen.

Listing 41.84: Text einfügen und löschen (Beispiel360.tcl)

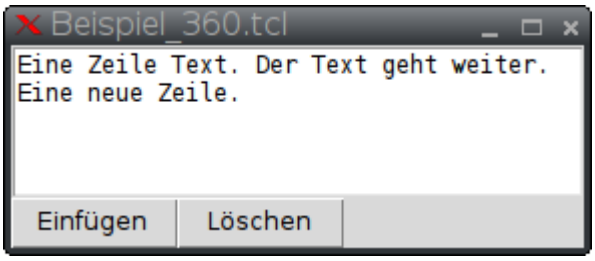
```
#!/usr/bin/env wish

text .tx -height 5 -width 40

.tx insert end "Eine Zeile Text."
.tx insert end " Der Text geht weiter."
.tx insert end "\nEine neue Zeile."

ttk::button .btEinfuegen -text "Einfuegen" -command {.tx insert 2.5 "zweite, "}
ttk::button .btLoeschen -text "Loeschen" -command {.tx delete 2.5 2.13}

pack .tx
pack .btEinfuegen -side left
pack .btLoeschen -side left
```



Nach einem Klick auf den Button `Einfügen`:



Nach einem Klick auf den Button `Löschen`:

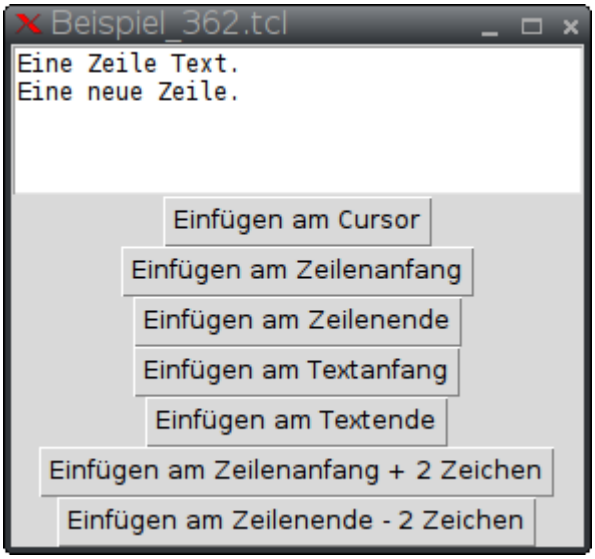


Das Text-Element verwaltet den Text über einen Index. Die erste Zeile hat den Index `1`, das erste Zeichen in einer Zeile hat den Index `0`. Somit bezeichnet der Index `2.5` das

sechste Zeichen in der zweiten Zeile. In Zeile 5 wird in das (noch leere) Text-Element ein Text am Ende hinzugefügt. In Zeile 6 wird erneut Text am Ende (das ist immer noch die 1. Zeile) hinzugefügt. In Zeile 7 wird mit dem Steuercode `\n` ein Zeilenumbruch eingefügt, so dass der weitere Text in Zeile 2 erscheint. In Zeile 9 wird ein Einfügen-Button definiert, der in der zweiten Zeile an sechster Stelle (= Index `5`) den Text `zweite,` einfügt. In Zeile 10 wird ein Löschen-Button definiert, der in der zweiten Zeile das sechste bis 14. Zeichen löscht. Das Einfügen von Text funktioniert nur, wenn das Text-Element im Status `normal` ist. Gegebenenfalls muss man den Status vor dem Einfügen des Textes ändern.

Listing 41.85: Text an verschiedenen Positionen einfügen (Beispiel362.tcl)

```
1  #!/usr/bin/env wish
2
3  text .tx -height 5 -width 40
4
5  .tx insert end "Eine Zeile Text."
6  .tx insert end "\nEine neue Zeile."
7
8  ttk::button .btEinfuegenCursor -text "Einfuegen am Cursor"
       -command {.tx insert insert "TEXT"}
9  ttk::button .btEinfuegenZeilenanfang -text "Einfuegen am
       Zeilenanfang" -command {.tx insert "insert linestart" "
       TEXT"}
10 ttk::button .btEinfuegenZeilenende -text "Einfuegen am
       Zeilenende" -command {.tx insert "insert lineend" "TEXT
       "}
11 ttk::button .btEinfuegenTextanfang -text "Einfuegen am
       Textanfang" -command {.tx insert 1.0 "TEXT"}
12 ttk::button .btEinfuegenTextende -text "Einfuegen am
       Textende" -command {.tx insert end "TEXT"}
13 ttk::button .btEinfuegenZeilenanfangPlus -text "Einfuegen
       am Zeilenanfang + 2 Zeichen" -command {.tx insert "
       insert linestart+2c" "TEXT"}
14 ttk::button .btEinfuegenZeilenendeMinus -text "Einfuegen
       am Zeilenende - 2 Zeichen" -command {.tx insert "insert
        lineend-2c" "TEXT"}
15
16 pack .tx
17 pack .btEinfuegenCursor
18 pack .btEinfuegenZeilenanfang
19 pack .btEinfuegenZeilenende
20 pack .btEinfuegenTextanfang
21 pack .btEinfuegenTextende
22 pack .btEinfuegenZeilenanfangPlus
23 pack .btEinfuegenZeilenendeMinus
```



Probieren Sie die verschiedenen Buttons aus.
Es gibt verschiedene Schlüsselwörter, die eine Textstelle bestimmen. Beachten Sie dabei die Anführungszeichen. Die Schlüsselwörter sind:

- `insert` = aktuelle Cursorposition
- `"insert linestart"` = Zeilenanfang
- `"insert lineend"` = Zeilenende
- `end` = Textende

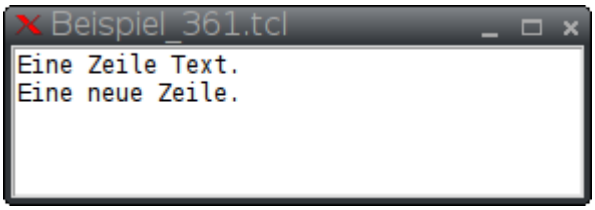
Außerdem kann man eine Anzahl an Zeichen zu der jeweiligen Position hinzuzählen oder abziehen:

- `"insert linestart+2c"` = 2 Zeichen nach dem Zeilenanfang
- `"insert lineend-2c"` = 2 Zeichen vor dem Zeilenende

Listing 41.86: Inhalt des Text-Elements in eine Variable speichern (Beispiel361.tcl)

```
#!/usr/bin/env wish

text .tx -height 5 -width 40

.tx insert end "Eine Zeile Text."
.tx insert end "\nEine neue Zeile."

pack .tx

set AnzahlZeilen [.tx count -lines 1.0 end]
puts "Zeilen: $AnzahlZeilen"

set AnzahlZeichen [.tx count -chars 1.0 1.end]
puts "Zeichen in 1. Zeile: $AnzahlZeichen"
```

```
15
16 set AnzahlZeichen [.tx count -chars 2.0 2.end]
17 puts "Zeichen in 2. Zeile: $AnzahlZeichen"
18
19 set AnzahlZeichen [.tx count -chars 1.0 end]
20 puts "Zeichen insgesamt: $AnzahlZeichen"
21
22 puts "Inhalt:"
23 set Text [.tx get 1.0 end]
24 puts $Text
```



In den Zeilen 5 bis 6 wird Text in das Text-Element eingefügt. In Zeile 10 wird die Anzahl der Zeilen ermittelt. In Zeile 13 bzw. in Zeile 16 wird die Anzahl der Zeichen in der ersten bzw. zweiten Zeile bestimmt. In Zeile 19 wird die gesamte Anzahl an Zeichen im Text-Element berechnet. Dabei wird immer das Zeilenende-Zeichen mitgezählt. In Zeile 23 wird der Inhalt des Text-Elements in die Variable Text gespeichert.

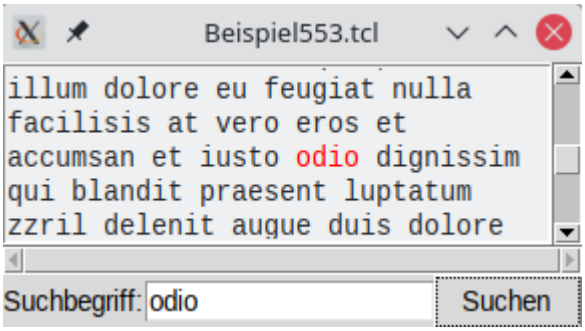
41.18.1 Text-Element durchsuchen

Das folgende Beispiel zeigt, wie man das Text-Element nach einem Suchbegriff durchsucht, die gefundenen Textstellen farbig markiert und die erste Textstelle zur Anzeige bringt.

Listing 41.87: Text-Element durchsuchen (Beispiel553.tcl)

```
1 #!/usr/bin/env wish
2
3 proc Suchen {Widget Text} {
4   # Durchsucht das Text-Widget nach einem Suchbegriff und ↩
      markiert die Treffer
5   $Widget tag remove Markierung 0.0 end
6   set Laenge [string length $Text]
7   set Liste [$Widget search -all $Text 1.0]
```

```
8   set Zaehler 0
9   set ErsterTreffer ""
10  foreach Element $Liste {
11    incr Zaehler
12    if {$Zaehler == 1} {
13      set ErsterTreffer $Element
14    }
15    set ZeileSpalte [split $Element "."]
16    lassign $ZeileSpalte Zeile Spalte
17    set Spalte [expr $Spalte + $Laenge]
18    $Widget tag add Markierung $Element [string cat $Zeile
        "." $Spalte]
19    $Widget tag configure Markierung -foreground red
20  }
21  if {$ErsterTreffer ne ""} {
22    $Widget see $ErsterTreffer
23  }
24 }
25
26 ttk::frame .frText
27 text .frText.tx -height 5 -width 15 -wrap word
    -xscrollcommand {.frText.sbX set} -yscrollcommand {
    .frText.sbY set}
28 ttk::scrollbar .frText.sbX -orient horizontal -command {
    .frText.tx xview}
29 ttk::scrollbar .frText.sbY -command {.frText.tx yview}
30
31 pack .frText.sbX -side bottom -fill x
32 pack .frText.sbY -side right -fill y
33 pack .frText.tx -side top -expand yes -fill both
34
35 ttk::frame .frSuchen
36 ttk::label .frSuchen.lbSuchen -text "Suchbegriff:"
37 ttk::entry .frSuchen.enSuchen -textvariable Suchbegriff
38 ttk::button .frSuchen.btSuchen -text "Suchen" -command {
    Suchen .frText.tx $Suchbegriff}
39
40 pack .frSuchen.lbSuchen -side left
41 pack .frSuchen.enSuchen -side left
42 pack .frSuchen.btSuchen -side left
43
44 pack .frText -side top -expand yes -fill both
45 pack .frSuchen -side bottom
```



In Zeile 5 werden alle (vorherigen) Markierungen entfernt. In Zeile 6 wird die Länge des Suchbegriffs gespeichert. In Zeile 7 wird das Text-Element durchsucht und alle gefundenen Textstellen in einer Liste gespeichert. Die gefundenen Textstellen haben die Form `Zeile.Spalte`. Mit der `foreach`-Schleife in den Zeilen 10 bis 20 werden alle Textstellen formatiert. In Zeile 18 wird die Textstelle mit einem `tag` mit dem Namen `Markierung` ausgezeichnet. In Zeile 19 wird die `Markierung` in der Farbe `red` formatiert.In Zeile 22 wird die erste Textstelle zur Anzeige gebracht.

41.19 Calendar

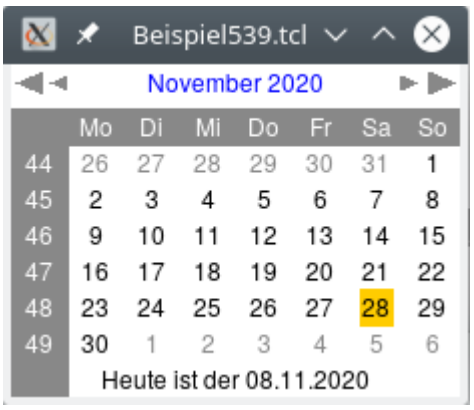
Das `widget::calendar`-Element zeigt einen Kalender an.

Tabelle 41.29: Die wichtigsten Optionen

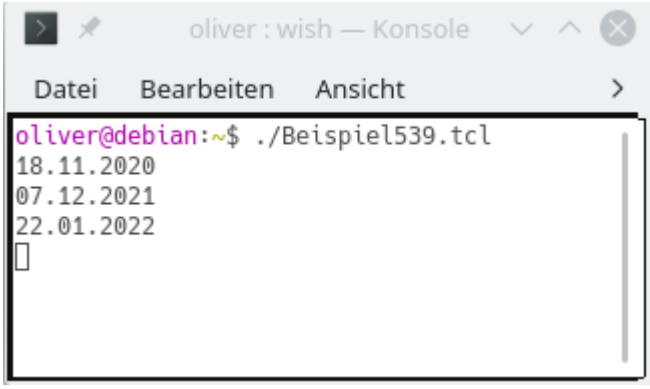
Option	Beschreibung
`-dateformat %d.%m.%Y`	Datumsformat (siehe Kapitel 7.2.1)
`-language de`	Sprache
`-textvariable Variablenname`	Name der Variable, die das Kalenderdatum enthält

Listing 41.88: Kalender (Beispiel539.tcl)

```
#!/usr/bin/env wish

package require widget::calendar

set Datum "28.11.2020" ; # muss zum Datumsformat passen
widget::calendar .cal -dateformat %d.%m.%Y -language de ↩
   -textvariable Datum
pack .cal
bind .cal <Double-ButtonPress-1> {puts $Datum}
```



Nach einem Doppelklick mit der linken Maustaste wird das Datum ausgewertet:



In Zeile 3 wird das Paket `widget::calendar` eingebunden. Das Paket ist Teil von `tklib`. In Zeile 5 wird eine Variable definiert, die das Kalenderdatum enthält. Wenn man den Kalender nicht mit dem aktuellen Datum vorbelegen möchte, kann man ein Datum selbst festlegen. Dabei muss das Datum in dem gleichen Format sein, wie im Kalender (Zeile 6) definiert. In Zeile 6 wird der Kalender definiert. In Zeile 7 wird der Kalender angezeigt. In Zeile 8 wird der Kalender mit dem Mausereignis `Doppelklick` verknüpft. Bei jedem Doppelklick wird die Variable `Datum` ausgewertet.

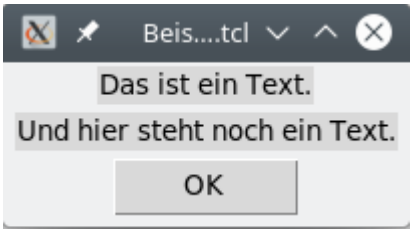
42 Weitere Elementeigenschaften

42.1 Einheitliche Hintergrundfarbe des Dialogs und der Elemente

Je nach Betriebssystem und Desktopumgebung kann es vorkommen, dass die Hintergrundfarbe des Dialogs und der Elemente unterschiedlich sind.

Listing 42.1: Unterschiedliche Hintergrundfarben (Beispiel527.tcl)

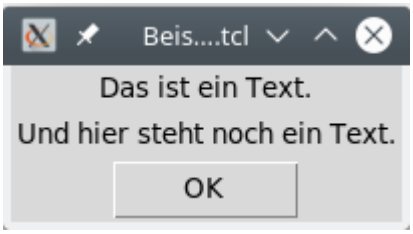
```
#!/usr/bin/env wish

ttk::label .lb1 -text "Das ist ein Text."
ttk::label .lb2 -text "Und hier steht noch ein Text."
ttk::button .bt -text "OK" -command {}
pack .lb1 -side top -padx 2 -pady 2
pack .lb2 -side top -padx 2 -pady 2
pack .bt -side top -padx 2 -pady 2
```



Wie man sieht, haben die Texte eine andere Hintergrundfarbe als der Dialog. Um die Hintergrundfarbe zu vereinheitlichen, sollten alle Elemente in ein `frame`-Element platziert werden.

Listing 42.2: Einheitliche Hintergrundfarben durch Frame-Element (Beispiel528.tcl)

```
#!/usr/bin/env wish

ttk::frame .fr
pack .fr -expand yes -fill both

ttk::label .fr.lb1 -text "Das ist ein Text."
ttk::label .fr.lb2 -text "Und hier steht noch ein Text."
ttk::button .fr.bt -text "OK" -command {}
pack .fr.lb1 -side top -padx 2 -pady 2
pack .fr.lb2 -side top -padx 2 -pady 2
pack .fr.bt -side bottom -padx 2 -pady 2
```



In den Zeilen 3 und 4 wird ein Rahmen erstellt, dessen Aufgabe es ist, alle Dialog-Elemente aufzunehmen. In den Zeilen 6 bis 8 werden alle Elemente in diesen Rahmen platziert.

42.2 Variablen und Befehle sind global gültig

Einige Elemente verfügen über Text- oder Listenvariablen bzw. können Befehle ausführen. So kann beispielsweise das `Label`-Element mit einer Textvariablen verknüpft werden, so dass sich der angezeigte Text automatisch ändert, wenn der Inhalt der Variablen geändert wird; ein `Listbox`-Element verwaltet die Einträge mit Hilfe einer Listenvariablen und ein `Button`-Element führt einen Befehl aus, wenn er angeklickt wird. Es ist sehr wichtig zu beachten, dass diese Variablen und Befehle immer global gültig sind, auch wenn die Elemente innerhalb einer Prozedur definiert werden. Im Folgenden sind ein paar Beispiele aufgeführt, die diese Thematik darstellen.

Listing 42.3: Eine Listbox mit drei Einträgen (Beispiel407.tcl)

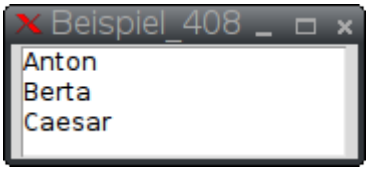
```
1 #!/usr/bin/env wish
2
3 set Liste {Anton Berta Caesar}
4 listbox .lb -listvariable Liste
5 pack .lb
```

In Zeile 3 wird eine Liste mit drei Einträgen festgelegt. In Zeile 4 wird die Listbox mit der Listen-Variablen verknüpft. In Zeile 5 wird die Listbox angezeigt. Das gesamte Programm läuft im globalen Namensraum.

Listing 42.4: Listbox in einer Prozedur mit globaler Variable (Beispiel408.tcl)

```
1 #!/usr/bin/env wish
2
3 proc Listbox {} {
4         set Liste {Dora Emil}
5         listbox .lb -listvariable Liste
6         pack .lb
7 }
8
```

```
9  set Liste {Anton Berta Caesar}
10 Listbox
```



Die Listbox wird in einer Prozedur erzeugt (Zeile 5). Die mit der Listbox verknüpfte Variable `Liste` gehört zum globalen Namensraum und ist nicht identisch mit der lokalen Variablen `Liste` aus der Prozedur (Zeile 4). Da die mit der Listbox verknüpfte Variable immer zum globalen Namensraum gehört, werden die Einträge gemäß Zeile 9 angezeigt.

Listing 42.5: Listbox in einer Prozedur zeigt keine Einträge an (Beispiel409.tcl)

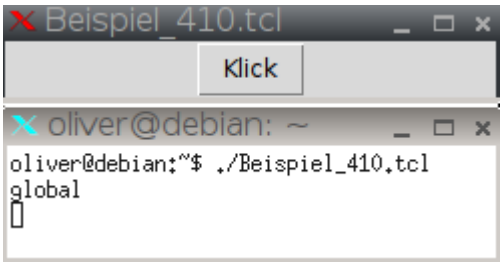
```
1 #!/usr/bin/env wish
2
3 proc Listbox {} {
4         set Liste {Dora Emil}
5         listbox .lb -listvariable Liste
6         pack .lb
7 }
8
9 Listbox
```

Die Liste ist leer, obwohl in Zeile 4 eine Variable Liste erzeugt und in Zeile 5 (scheinbar) mit der Listbox verknüpft wurde. In Wahrheit ist aber die Variable `Liste` aus Zeile 4 nur lokal gültig und die mit der Listbox verknüpfte Variable `Liste` gehört zum globalen Namensraum. Dort wurde aber die Variable nicht mit Einträgen belegt, so dass sie leer ist. Diese Fehlerquelle ist tückisch, da es bei der Programmausführung zu keiner Fehlermeldung kommt.

Listing 42.6: Command-Befehle werden im globalen Namensraum ausgeführt (Beispiel410.tcl)

```
1 #!/usr/bin/env wish
2
3 proc Button {} {
4         set Text "lokal"
5         ttk::button .bt -text "Klick" -command {puts $Text}
6         pack .bt
7 }
```

```
8
9  set Text "global"
10 Button
```



In Zeile 9 wird eine Variable `Text` definiert. In Zeile 10 wird die Prozedur `Button` aufgerufen. In Zeile 4 wird ebenfalls eine Variable `Text` definiert. Diese ist aber lokal gültig und damit trotz gleichem Namen unterschiedlich zur Variablen `Text` aus Zeile 9. In Zeile 5 wird ein Button erzeugt, der beim Anklicken den Inhalt der Variablen `Text` anzeigt. Man erkennt, dass die Befehle des `command`-Befehls im globalen Namensraum ausgeführt werden, obwohl das `Button`-Element in einer Prozedur erzeugt wurde. Denn es wird die Variable `Text` aus Zeile 9 angezeigt und nicht die Variable `Text` aus Zeile 4 in der Prozedur. Außerdem sieht man, dass der `command`-Befehl auf die globale Variable `Text` zugreift, ohne dass in der Prozedur die Variable mit dem Befehl `global` als global festgelegt wurde.

42.3 Farben

Optionen:

- `-foreground Farbe1`
- `-background Farbe2`

Bei den klassischen Elementen legen die Optionen `-foreground` und `-background` die Vorder- und Hintergrundfarbe fest. Bei z. B. einem Label-Element steht die Vordergrundfarbe für die Textfarbe.

Die beiden Optionen funktionieren auch mit einigen ttk-Elementen, sollten dort aber möglichst nicht verwendet werden. Bei ttk-Elementen definiert man besser einen Stil und ordnet diesen Stil dem ttk-Element zu.

Es gibt sehr viele Farben, die direkt über ihren Namen angesprochen werden können, z. B. `SeaGreen2`. Die Groß-/Kleinschreibung ist dabei nicht wichtig. Leider gibt es aber keine Liste aller Farbnamen mit deren RGB-Werten. Deshalb ist es meistens besser, statt des Farbnamens den RGB-Wert anzugeben.

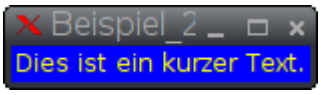
Listing 42.7: Label mit farbigem Text (Farbname) (Beispiel225.tcl)

```
1 #!/usr/bin/env wish
2
3 label .lb -text "Dies ist ein kurzer Text." -foreground ↩
    Gold -background DodgerBlue
4 pack .lb
```

Man kann die Farbe auch als RGB-Wert angeben. Dann muss man ein # Zeichen voranstellen. Der RGB-Wert ist hexadezimal anzugeben (Format `#RRGGBB`). Dabei steht RR für Rot, GG für Gelb und BB für Blau.

Listing 42.8: Label mit RGB-Farbe (hexadezimal) (Beispiel226.tcl)

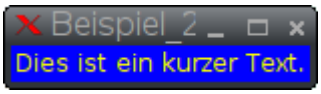
```
#!/usr/bin/env wish

label .lb -text "Dies ist ein kurzer Text." -foreground #FFFF00 -background #0000FF
pack .lb
```



Man kann die RGB-Werte auch dezimal übergeben, wie das folgende Beispiel zeigt.

Listing 42.9: Label mit RGB-Farbe (dezimal) (Beispiel227.tcl)

```
1 #!/usr/bin/env wish
2
3 label .lb -text "Dies ist ein kurzer Text." -foreground [format "#%02x%02x%02x" 255 255 0] -background [format "#%02x%02x%02x" 0 0 255]
4 pack .lb
```



Durch die `format`-Befehle in Zeile 3 werden die dezimalen RGB-Werte (`255 255 0`) bzw. (`0 0 255`) in hexadezimale Werte umgewandelt.

42.4 Schrift

Option:

- `-font "Schriftart Schriftgröße Schriftstil"`

Bei den klassischen Elementen (also nicht die ttk-Elemente) wird die Schrift direkt beim Element festgelegt. Die Option `-font` definiert die Schriftart (z. B. Times, Helvetica, Courier), Schriftgröße und den Schriftstil. Es dürfen mehrere Schriftstile kombiniert werden (z. B. fett und unterstrichen).

Die Option funktioniert auch mit einigen ttk-Elementen, sollte dort aber möglichst nicht verwendet werden. Bei ttk-Elementen definiert man besser einen Stil und ordnet diesen Stil dem ttk-Element zu.

Tabelle 42.1: Schriftart

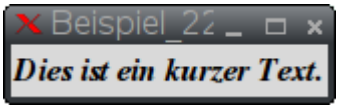
Schriftart	Beschreibung
`Helvetica`	Eine serifenlose Schrift
`Times`	Eine Serifen-Schrift
`Courier`	Eine nichtproportionale Schrift

Tabelle 42.2: Schriftstile

Schriftstil	Beschreibung
`bold`	fett
`italic`	kursiv
`underline`	unterstrichen
`overstrike`	durchgestrichen
`normal`	normal
`roman`	roman

Listing 42.10: Label mit Schriftart Times in 12 Punkt-Größe, fett und kursiv (Beispiel228.tcl)

```
#!/usr/bin/env wish

label .lb -text "Dies ist ein kurzer Text." -font "Times 12 bold italic"
pack .lb
```



42.5 Bitmaps (nur bei klassischen Elementen)

Option:

- `-bitmap @Bild.xbm -compound Ausrichtung -foreground Farbe1 -background Farbe2`

Einige Elemente können Bitmaps anzeigen. Bitmaps werden mit der Option `-bitmap` festgelegt. Dem Dateiname des Bitmaps muss man ein `@` Zeichen vorangestellt werden. Bitmap-Dateien müssen das Dateiformat `xbm` haben. Ein Bitmap ist ein Muster von gesetzten und nicht gesetzten Pixeln. Die Farbe des Bitmaps wird eingestellt mit den Optionen `-foreground` und `-background`. Die Farbe ist optional. Die Ausrichtung des

Bitmaps innerhalb des Elements erfolgt mit der Option `-compound`. Zulässige Werte sind `top`, `bottom`, `left`, `right` und `center`.

Listing 42.11: Label mit Bitmap (Beispiel229.tcl)

```
#!/usr/bin/env wish

label .lb -text "Text" -bitmap @Bild.xbm -compound left
pack .lb -side left
```

Listing 42.12: Label mit Bitmap und Farbdefinition (Beispiel230.tcl)

```
#!/usr/bin/env wish

label .lb -text "Text" -bitmap @Bild.xbm -compound left ↲
   -foreground #FFFF00 -background #0000FF
pack .lb -side left
```

42.6 Bilder

Befehl:

- `image create photo Bildname -file DateinameDesBildes`

Option:

- `-image Bildname -compound left`

Einige Elemente können Bilder anzeigen. Bilder sind in Tcl/Tk Objekte, die mit dem Befehl `image create` aus einer Bilddatei erstellt werden und mit der Option `-image` einem Element zugewiesen werden. Bild-Dateien dürfen das Dateiformat `xbm`, `xpm`, `ppm`, `gif` und (ab Tcl 8.6) auch `png` haben. Wenn man das Paket `libtk-img` installiert hat, kann man auch `jpg`-Dateien verwenden. Falls man Bilder bearbeiten möchte (z. B. Größe ändern), ist es (unter Linux) hilfreich imagemagick zu installieren und dessen Befehle zu benutzen (siehe späteres Beispiel).

Listing 42.13: Label mit Bild (Beispiel231.tcl)

```
#!/usr/bin/env wish

image create photo MeinFoto -file MeinFoto.ppm
```

```
4 ttk::label .lb -text "Eine schoene Windmuehle" -image ↩
    MeinFoto -compound left
5 pack .lb -side left
```

In Zeile 3 wird das Bildobjekt `MeinFoto` aus der Bilddatei `MeinFoto.ppm` erzeugt. In Zeile 4 wird das Bildobjekt dem Label zugewiesen.

Listing 42.14: jpg-Datei verwenden (libtk-img benutzen) (Beispiel364.tcl)

```
1 #!/usr/bin/env wish
2
3 package require Img
4
5 image create photo MeinFoto -file MeinFoto.jpg
6 ttk::label .lb -text "Eine schoene Windmuehle" -image ↩
    MeinFoto -compound left
7 pack .lb -side left
```

Damit man `jpg`-Dateien verwenden kann, muss das Paket `libtk-img` installiert sein. Alternativ bieten die Linux-Distributionen die Installation des Pakets an. In Zeile 3 wird das Paket `Img` eingebunden (bitte die Großschreibung beachten). In Zeile 5 wird eine `jpg`-Datei als Foto verwendet.
Wenn man aus Tcl/Tk heraus Bilder verändern will (z. B. die Bildgröße), kann man das Paket Imagemagick verwenden. Das folgende Beispiele verkleinert alle Bilder im aktuellen Ordner auf die Größe 300x300 Pixel.

Listing 42.15: Imagemagick benutzen (Beispiel365.tcl)

```
1 #!/usr/bin/env tclsh
2
3 exec mogrify -resize 300x300 *.JPG
```

In Zeile 3 ruft der Befehl `exec` den Befehl `mogrify` (aus dem Paket imagemagick) auf. Imagemagick verkleinert dann die Bilder.

42.7 Prüfen, ob ein Element existiert

Befehl:

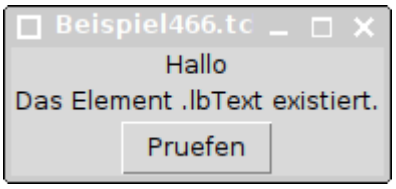
- `[winfo exists Elementname]`

Mit dem Befehl `[winfo exists Elementname]` kann man prüfen, ob ein bestimmtes Element existiert. Der Befehl gibt den Wert 0 oder 1 zurück.

Listing 42.16: Prüfen, ob ein Element existiert (Beispiel466.tcl)

```
1 #!/usr/bin/env wish
2
3 proc Pruefen {} {
4         global Ergebnis
5         if {[winfo exists .lbText]} {
```

```
6                   set Ergebnis "Das Element .lbText ↩
                       existiert."
7           } else {
8                   set Ergebnis "Das Element .lbText ↩
                       existiert nicht."
9           }
10          update idletasks
11  }
12
13  set Ergebnis ""
14  ttk::label .lbText -text "Hallo"
15  ttk::label .lbErgebnis -textvariable Ergebnis
16  ttk::button .btPruefen -text "Pruefen" -command {Pruefen}
17  pack .lbText
18  pack .lbErgebnis
19  pack .btPruefen
```



In Zeile 5 wird geprüft, ob das Element `.lbText` existiert.

42.8 Größe und Position eines Elements abfragen

Befehl:

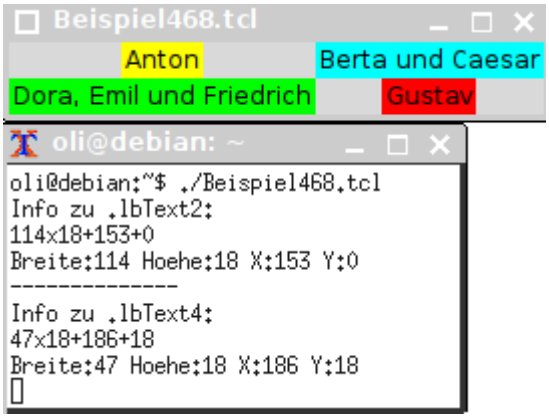
- `[winfo geometry Elementname]`

Mit dem Befehl `[winfo geometry Elementname]` kann man die Größe und die Position eines Elements abfragen. Die Angabe ist in der Form `BreitexHöhe+X+Y`.

Listing 42.17: Größe und Position eines Elements abfragen (Beispiel468.tcl)

```
1  #!/usr/bin/env wish
2
3  catch {console show}
4
5  ttk::label .lbText1 -text "Anton" -background #FFFF00
6  ttk::label .lbText2 -text "Berta und Caesar" -background ↩
      #00FFFF
7  ttk::label .lbText3 -text "Dora, Emil und Friedrich" ↩
      -background #00FF00
8  ttk::label .lbText4 -text "Gustav" -background #FF0000
9  grid .lbText1 -row 0 -column 0
10 grid .lbText2 -row 0 -column 1
11 grid .lbText3 -row 1 -column 0
12 grid .lbText4 -row 1 -column 1
13 update idletasks
```

```
14
15 set Info [winfo geometry .lbText2]
16 puts "Info zu .lbText2:"
17 puts "$Info"
18 set Liste [split $Info "x+"]
19 lassign $Liste Breite Hoehe X Y
20 puts "Breite:$Breite Hoehe:$Hoehe X:$X Y:$Y"
21 puts "--------------"
22 set Info [winfo geometry .lbText4]
23 puts "Info zu .lbText4:"
24 puts "$Info"
25 set Liste [split $Info "x+"]
26 lassign $Liste Breite Hoehe X Y
27 puts "Breite:$Breite Hoehe:$Hoehe X:$X Y:$Y"
```



In den Zeilen 15 und 22 wird die Größe und Position der beiden Elemente `.lbText2` bzw. `.lbText4` abgefragt.

42.9 Element löschen

Befehl:

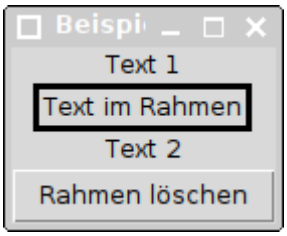
- `destroy Elementname`

Der Befehl `destroy Elementname` löscht ein Element inklusive aller Unterelemente. Das Element sowie seine Unterelemente existieren danach nicht mehr.

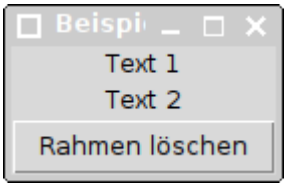
Listing 42.18: Element löschen (Beispiel469.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::label .lb1 -text "Text 1"
4 ttk::frame .fr -borderwidth 3 -relief solid
5 ttk::label .fr.lb3 -text "Text im Rahmen"
6 ttk::label .lb2 -text "Text 2"
7 ttk::button .bt -text "Rahmen loeschen" -command {destroy ↩
    .fr}
8 pack .lb1
```

```
9  pack .fr.lb3
10 pack .fr
11 pack .lb2
12 pack .bt
```



Nach einem Klick auf den Button `Rahmen löschen`:



In Zeile 7 wird mit dem Befehl `destroy .fr` das Rahmen-Element `.fr` gelöscht. Damit wird zugleich auch das Unterelement `.fr.lb3` gelöscht.

42.10 Element deaktivieren und aktivieren

Befehle:

- `Elementname configure -state normal`
- `Elementname configure -state disabled`
- `Elementname configure -state readonly`

Die Option `-state` legt fest, ob ein Element (z. B. eine Eingabefeld oder ein Button) aktiv ist und vom Anwender bearbeitet oder angeklickt werden kann. Der Zustand `readonly` ist nur bei einigen Elementen (z. B. beim `entry`-Element) verfügbar. Beachten Sie auch, dass der Status beim `Scale`-Element auf andere Art gesetzt wird (siehe Kapitel 41.13 auf Seite 431).

Tabelle 42.3: Zulässige Zustände eines Elements

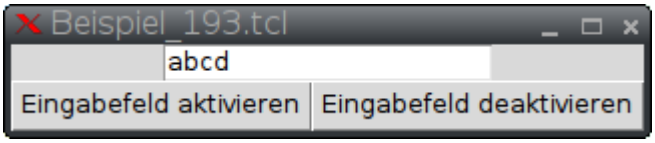
Zustand	Beschreibung
`-state normal`	Element ist aktiv
`-state disabled`	Element nicht aktiv

Tabelle 42.3: Zulässige Zustände eines Elements

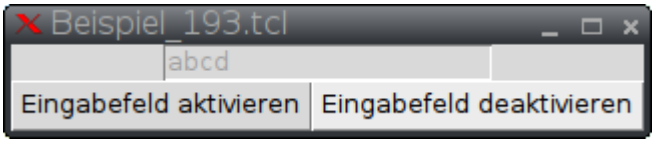
Zustand	Beschreibung
`-state readonly`	Element ist nicht aktiv, der Inhalt kann aber selektiert und kopiert werden

Listing 42.19: Element aktivieren und deaktivieren (Beispiel193.tcl)

```
#!/usr/bin/env wish

ttk::entry .en
ttk::button .btAktivieren -text "Eingabefeld aktivieren" -command {.en configure -state normal}
ttk::button .btDeaktivieren -text "Eingabefeld deaktivieren" -command {.en configure -state disabled}

pack .en -side top
pack .btAktivieren -side left
pack .btDeaktivieren -side left
```



Durch einen Klick auf den Button `Deaktivieren` kann man das Eingabefeld deaktivieren.



Beim `ttk::scale`-Element muss man beachten, dass der Status nicht wie bei den anderen Elementen mit `configure Elementname -state disabled` gesetzt wird, sondern mit `Elementname state disabled` (ohne `configure`und ohne das Minuszeichen vor `state`). Darüber hinaus funktioniert das Zurücksetzen in den `normal`-Zustand nicht mit `Elementname state normal`, sondern mit `Elementname state !disabled`.

Listing 42.20: Scale-Element aktivieren und deaktivieren (Beispiel502.tcl)

```
#!/usr/bin/env wish

ttk::scale .sc -orient horizontal -from 0 -to 100 -length 200 -variable Wert
ttk::button .btNormal -text "Normal" -command {.sc state !disabled}
```

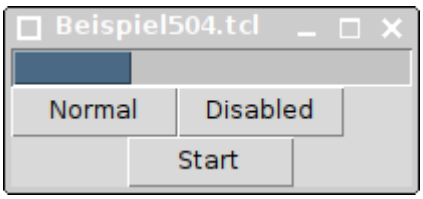
```
5 ttk::button .btDisabled -text "Disabled" -command {.sc ↩
    state disabled}
6
7 pack .sc -side top
8 pack .btNormal -side left
9 pack .btDisabled -side left
```

Die gleiche Besonderheit gilt auch für das `ttk::progressbar`-Element.

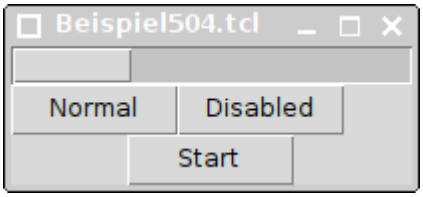
Listing 42.21: Progressbar-Element aktivieren und deaktivieren (Beispiel504.tcl)

```
1 #!/usr/bin/env wish
2
3 proc Schleife {Element} {
4   for {set Zaehler 0} {$Zaehler <= 30} {incr Zaehler} {
5     after 30
6     $Element configure -value $Zaehler
7     update idletask
8   }
9 }
10
11 set Wert 0
12 ttk::progressbar .pb -orient horizontal -maximum 100 ↩
    -length 200 -value 0
13 ttk::button .bt -text "Start" -command {Schleife .pb}
14 pack .pb -side top
15 pack .bt -side bottom
16
17 ttk::button .btNormal -text "Normal" -command {.pb state !↩
    disabled}
18 ttk::button .btDisabled -text "Disabled" -command {.pb ↩
    state disabled}
19 pack .btNormal -side left
20 pack .btDisabled -side left
```

Im `normal`-Zustand sieht der Fortschrittsbalken wie folgt aus:



Und so sieht er im `disbabled`-Zustand aus:



42.11 Element ausblenden

Befehle:

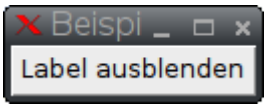
- `pack forget Elementname`
- `grid forget Elementname`
- `grid remove Elementname`

Abhängig vom verwendeten Geometriemanager kann man mit dem Befehl `pack forget` bzw. `grid forget` / `grid remove` ein Element ausblenden. Das Element existiert weiterhin, wird aber nicht angezeigt. Der Unterschied zwischen `grid forget` und `grid remove` liegt darin, dass bei `grid forget` die Konfiguration des Elements verloren geht und bei einem späteren Einblenden des Elements wieder die Initialkonfiguration gesetzt wird. Demgegenüber wird bei `grid remove` die eingestellte Konfiguration beibehalten.

Listing 42.22: Ausblenden mit pack forget (Beispiel194.tcl)

```
#!/usr/bin/env wish

ttk::label .lb -text Hallo
ttk::button .bt -text "Label ausblenden" -command {pack forget .lb}
pack .lb -side top
pack .bt -side bottom
```

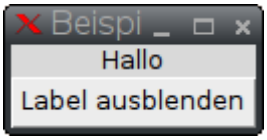
Wenn man auf den Button `Label ausblenden` klickt, wird das Label ausgeblendet.



Listing 42.23: Ausblenden mit grid forget (Beispiel417.tcl)

```
#!/usr/bin/env wish

ttk::label .lb -text Hallo
```

```
ttk::button .bt -text "Label ausblenden" -command {grid forget .lb}
grid .lb -row 0 -column 0
grid .bt -row 1 -column 0
```



Wenn man auf den Button `Label ausblenden` klickt, wird das Label ausgeblendet.

42.12 Element einblenden

Befehle:

- `pack Elementname`
- `grid Elementname`

Mit dem Befehl `pack` bzw. `grid` kann man ein ausgeblendetes Element wieder anzeigen.

Listing 42.24: Einblenden mit pack (Beispiel195.tcl)

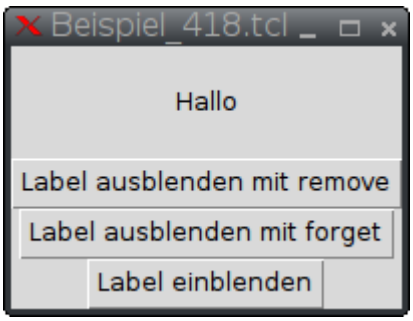
```
#!/usr/bin/env wish

ttk::frame .fr
ttk::label .fr.lb -text Hallo
ttk::button .btAusblenden -text "Label ausblenden" -command {pack forget .fr.lb}
ttk::button .btEinblenden -text "Label einblenden" -command {pack .fr.lb}

pack .fr -side top
pack .fr.lb -side top
pack .btAusblenden -side left
pack .btEinblenden -side left
```

Nach einem Klick auf den Button `Label ausblenden`:

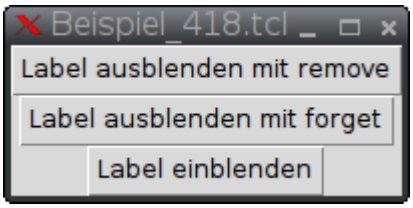
Nach einem Klick auf den Button `Label einblenden`:

Listing 42.25: Einblenden mit grid (Beispiel418.tcl)

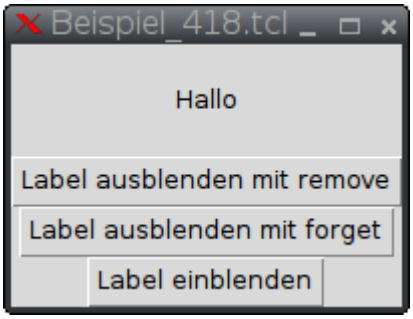
```
#!/usr/bin/env wish

ttk::label .lb -text Hallo
ttk::button .btAusblendenRemove -text "Label ausblenden mit remove" -command {grid remove .lb}
ttk::button .btAusblendenForget -text "Label ausblenden mit forget" -command {grid forget .lb}
ttk::button .btEinblenden -text "Label einblenden" -command {grid .lb -row 0 -column 0}

grid .lb -row 0 -column 0 -pady 20
grid .btAusblendenRemove -row 1 -column 0
grid .btAusblendenForget -row 2 -column 0
grid .btEinblenden -row 3 -column 0
```



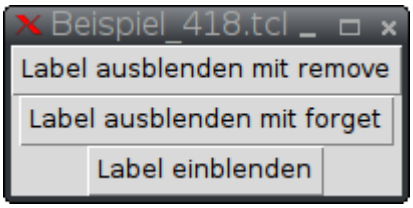
Nach dem Klick auf den Button `Label ausblenden mit remove`.



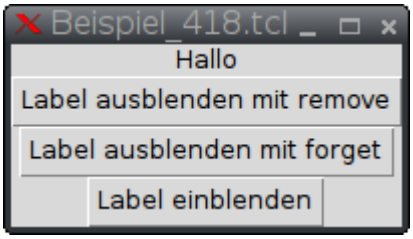
Nach dem Klick auf den Button `Label einblenden` erscheint das Label-Element wieder mit seiner eingestellten Konfiguration (vertikaler Abstand `-pady 20`).



Nach dem Klick auf den Button `Label ausblenden mit forget`.



Nach dem Klick auf den Button `Label einblenden` erscheint das Label-Element mit seiner Initial-Konfiguration, d. h. der vertikale Abstand (`-pady 20`) gilt nicht mehr.



42.13 Element verschieben

Befehle:

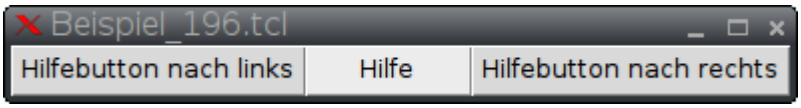
- `pack forget Elementname`
- `pack Elementname -before AnderesElement`
- `pack Elementname -after AnderesElement`

Wenn man ein Element an eine andere Stelle verschieben möchte, blendet man das Element mit dem Befehl `pack forget` zuerst aus. Danach blendet man das Element mit dem Befehl `pack` und der Option `-before` oder `-after` an einer neuen Stelle wieder ein.

Listing 42.26: Element verschieben (Beispiel196.tcl)

```
#!/usr/bin/env wish

ttk::button .btHilfe -text Hilfe
ttk::button .btLinks -text "Hilfebutton nach links" ↲
   -command {pack forget .btHilfe; pack .btHilfe -before ↲
   .btLinks -side left}
```

```
5 ttk::button .btRechts -text "Hilfebutton nach rechts" ↩
    -command {pack forget .btHilfe; pack .btHilfe -after ↩
    .btRechts -side left}
6
7 pack .btLinks -side left
8 pack .btHilfe -side left
9 pack .btRechts -side left
```



42.14 Fokus auf ein Element setzen

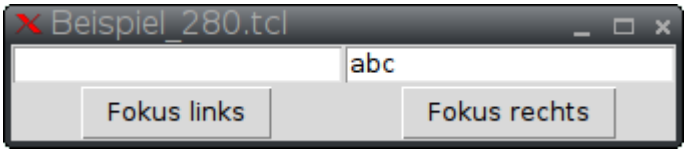
Befehle:

- `focus Elementname`
- `set Variable [focus]`

Der Befehl `focus` setzt den (Eingabe-) Fokus auf ein Element. Der Befehl `[focus]` gibt den Namen des Elements zurück, das aktuell den Fokus hat.

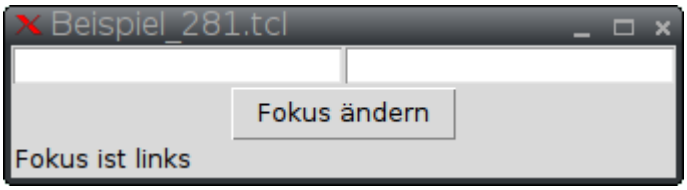
Listing 42.27: Fokus setzen (Beispiel280.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::entry .en1
4  ttk::entry .en2
5  ttk::button .btFokusLinks -text "Fokus links" -command {↩
     focus .en1}
6  ttk::button .btFokusRechts -text "Fokus rechts" -command {↩
     focus .en2}
7
8  grid .en1 -row 0 -column 0
9  grid .en2 -row 0 -column 1
10 grid .btFokusLinks -row 1 -column 0
11 grid .btFokusRechts -row 1 -column 1
```

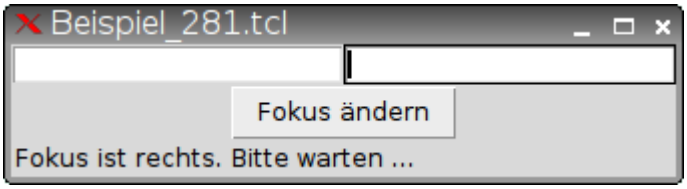


Listing 42.28: Fokus speichern und ändern (Beispiel281.tcl)

```
#!/usr/bin/env wish

proc FokusAendern {} {
  set FokusAlt [focus]
  focus .en2
  .lb configure -text "Fokus ist rechts. Bitte warten ..."
  update idletasks
  after 3000
  focus $FokusAlt
  .lb configure -text "Fokus ist links"
  update idletasks
}

ttk::entry .en1
ttk::entry .en2
ttk::button .btFokus -text "Fokus aendern" -command ↩
   FokusAendern
ttk::label .lb

grid .en1 -row 0 -column 0
grid .en2 -row 0 -column 1
grid .btFokus -row 1 -column 0 -columnspan 2
grid .lb -row 2 -column 0 -columnspan 2 -sticky w

focus .en1
.lb configure -text "Fokus ist links"
```



Nach einem Klick auf den Button `Fokus ändern`, wird der Fokus für drei Sekunden auf das rechte Eingabefeld gesetzt:



In Zeile 24 wird der Fokus beim Programmstart auf das linke Eingabefeld gesetzt. In Zeile 4 wird der aktuelle Fokus gespeichert. In Zeile 5 wird der Fokus auf das rechte Eingabefeld gesetzt. In Zeile 7 wird die grafische Darstellung aktualisiert. In Zeile 8 wartet das Programm drei Sekunden. In Zeile 9 wird der ursprüngliche Fokus wieder hergestellt. In Zeile 11 wird die grafische Darstellung aktualisiert.

42.15 Eigenschaften eines Elements abfragen und ändern

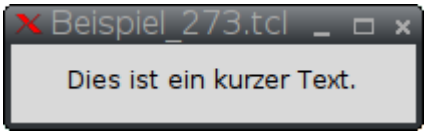
Befehle:

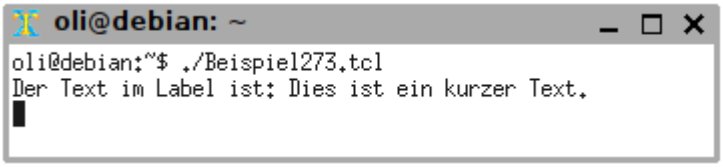
- `Elementname cget Option`
- `Elementname configure Option`

Mit dem Befehl `cget` kann man die Eigenschaften eines Elements abfragen. Mit dem `configure`-Befehl kann man die Eigenschaften der meisten Elemente ändern. Bei einigen Elementen (zum Beispiel Menü oder Notebook) werden die Eigenschaften auf andere Art geändert (siehe die dortigen Beispiele).

Listing 42.29: Eigenschaft eines Elementen abfragen (Beispiel273.tcl)

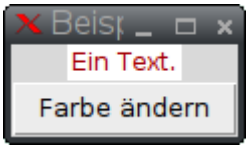
```
1 #!/usr/bin/env wish
2
3 ttk::label .lb -text "Dies ist ein kurzer Text."
4 pack .lb
5 set Text [.lb cget -text]
6 puts "Der Text im Label ist: $Text"
```





Listing 42.30: Eigenschaft eines Elements ändern (Beispiel274.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::label .lb -text "Ein Text." -background #A00000 ↩
    -foreground #FFFFFF
4 ttk::button .bt -text "Farbe aendern" -command {.lb ↩
    configure -background #FFFFFF -foreground #A00000}
5
6 pack .lb -side top
7 pack .bt -side top
```

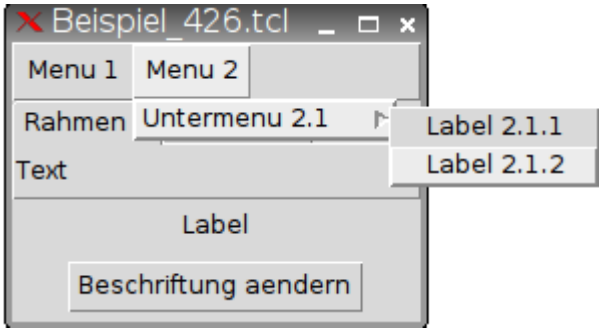


Das folgende Programm zeigt für mehrere Elemente, wie man deren Eigenschaften ändern kann.

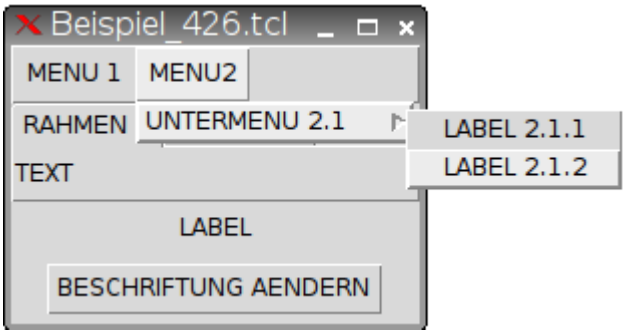
Listing 42.31: Viele verschiedene Elemente ändern (Beispiel426.tcl)

```
#!/usr/bin/env wish

proc BeschriftungAendern {} {
        .lb configure -text "LABEL"
        .bt configure -text "BESCHRIFTUNG AENDERN"

        .n tab .n.f1 -text "RAHMEN 1"
        .n tab .n.f2 -text "RAHMEN 2"
        .n.f1.lb configure -text "TEXT"

        .mbar entryconfigure 1 -label "MENU 1"
        .mbar entryconfigure 2 -label "MENU2"

        .mbar.menu1 entryconfigure 0 -label "UNTERMENU 1.1"
        .mbar.menu2 entryconfigure 0 -label "UNTERMENU 2.1"

        .mbar.menu2.menu21 entryconfigure 0 -label "LABEL 2.1.1"
        .mbar.menu2.menu21 entryconfigure 1 -label "LABEL 2.1.2"
}

menu .mbar
. configure -menu .mbar
.mbar add cascade -label "Menu 1" -menu .mbar.menu1
.mbar add cascade -label "Menu 2" -menu .mbar.menu2
menu .mbar.menu1 -tearoff 0
.mbar.menu1 add command -label "Untermenu 1.1"
menu .mbar.menu2 -tearoff 0
.mbar.menu2 add cascade -label "Untermenu 2.1" -menu .mbar.menu2.menu21
menu .mbar.menu2.menu21 -tearoff 0
.mbar.menu2.menu21 add radiobutton -label "Label 2.1.1"
.mbar.menu2.menu21 add radiobutton -label "Label 2.1.2"

```

```
33 ttk::label .lb -text "Label"
34 ttk::button .bt -text "Beschriftung aendern" -command {
      BeschriftungAendern}
35
36 ttk::notebook .n
37 ttk::frame .n.f1
38 ttk::frame .n.f2
39 .n add .n.f1 -text "Rahmen 1"
40 .n add .n.f2 -text "Rahmen 2"
41
42 ttk::label .n.f1.lb -text "Text"
43
44 pack .n -side top -fill both -expand 1
45 pack .n.f1.lb -side top -anchor w -pady 5
46 pack .lb -pady 5
47 pack .bt -pady 5
```



Nach dem Klick auf den Button `Beschriftung ändern` wird alles in Großbuchstaben dargestellt:



In den Zeilen 4 und 5 werden die Beschriftung von Label und Button mit dem Befehl `configure` geändert. In den Zeilen 7 und 8 wird die Beschriftung des Notebooks geändert. Dies erfolgt in dem man den Text des Reiters (`tab`) ändert. In den Zeile 11 und 12 wird die oberste Ebene des Menüs geändert. Hinter dem Befehl `entryconfigure` folgt der Index des Menüeintrags. Der erste Menüentrag hat den Index 1. In den Zeilen 14 bis 18 werden die Untermenüs geändert. Das erste Untermenü hat den Index 0.

42.16 Tabulatorreihenfolge der Elemente

Befehle:

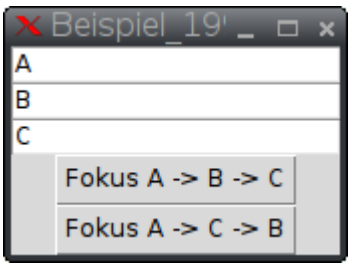
- `raise Elementname`
- `lower Elementname`
- `raise Elementname NachEinemAnderenElement`
- `lower Elementname VorEinemAnderenElement`

Die Tabulatorreihenfolge ist gleich der Reihenfolge, in der die Elemente erzeugt wurden. Wenn man die Tabulatorreihenfolge ändern will (aber nicht die Reihenfolge, in der die Elemente erzeugt werden), nimmt man die Befehle `raise` bzw. `lower`. Man kann sich die Elemente aufeinander gestapelt vorstellen. Das erste erzeugte Element liegt ganz unten und ist das erste Element in der Tabulatorreihenfolge. Das zweite Element wird darauf platziert und ist das zweite Element in der Tabulatorreihenfolge. Mit dem Befehl `raise` platziert man ein Element auf dem Stapel (es kommt also als letztes Element in der Tabulatorreihenfolge dran), mit dem Befehl `lower` schiebt man ein Element unter den Stapel (es wird also zum ersten Element).

Listing 42.32: Tabulatorreihenfolge ändern (Beispiel199.tcl)

```
#!/usr/bin/env wish

proc FokusABC {} {
  raise .enA
  raise .enB
  raise .enC
  raise .btABC
  raise .btACB
}

proc FokusACB {} {
  raise .enA
  raise .enC
  raise .enB
  raise .btABC
  raise .btACB
}

ttk::entry .enA
ttk::entry .enB
ttk::entry .enC
ttk::button .btABC -text "Fokus A -> B -> C" -command FokusABC
ttk::button .btACB -text "Fokus A -> C -> B" -command FokusACB
focus .enA
pack .enA
pack .enB
pack .enC
```

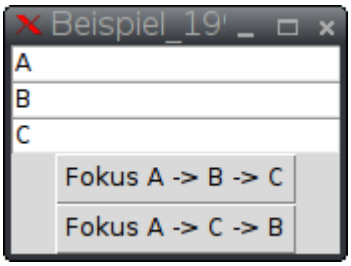
```
28 pack .btABC
29 pack .btACB
```



Zum Ausprobieren klicken Sie mit der Tabulatortaste durch das Fenster und auf die Buttons.
In den Zeilen 19 bis 21 werden drei Eingabefelder erzeugt. Damit wird zugleich die Tabulatorreihenfolge bestimmt. Wenn man den Button `A->C->B` anklickt, wird die Tabulatorreihenfolge geändert.

Listing 42.33: Tabulatorreihenfolge für ein Element ändern (Beispiel200.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::entry .enA
4  ttk::entry .enB
5  ttk::entry .enC
6  ttk::button .btABC -text "Fokus A -> B -> C" -command {
       lower .enB .enC}
7  ttk::button .btACB -text "Fokus A -> C -> B" -command {
       raise .enB .enC}
8  focus .enA
9  pack .enA
10 pack .enB
11 pack .enC
12 pack .btABC
13 pack .btACB
```



In der Zeile 7 wird das Element B hinter das Element C einsortiert (das Element B wird sozusagen auf das Element C gelegt). In der Zeile 6 wird das Element B vor das Element C einsortiert (das Element B wird unter das Element C gelegt).

42.17 Automatische Größenanpassung abschalten

Befehl:

- `pack propagate Elementname 0`

Normalerweise wird eine Element (z. B. ein Rahmen) genau so groß dargestellt wie nötig. Das bedeutet, dass das Element automatisch soweit geschrumpft wird, bis es genau um die ihm untergeordneten Elemente passt. Mit dem Befehl `pack propagate Elementname 0` kann man die automatische Größenanpassung abschalten.

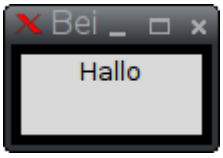
Listing 42.34: Rahmen verändert automatisch die Größe (Beispiel197.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::frame .frRahmen -borderwidth 5 -relief solid -width ↩
    100 -height 50
4 ttk::label .frRahmen.lbLabel -text "Hallo"
5 pack .frRahmen
6 pack .frRahmen.lbLabel
```

Obwohl in Zeile 3 die Größe des Rahmens festgelegt wurde, wird der Rahmen automatisch verkleinert, nachdem das Label in den Rahmen eingefügt wurde. Da der Rahmen nur als Container für andere Elemente dient, passt sich die Rahmengröße automatisch an den Inhalt an. Die Optionen `-width` und `-height` werden nicht mehr beachtet, sobald weitere Elemente in dem Rahmen platziert werden.

Listing 42.35: Rahmen behält seine Größe (Beispiel198.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::frame .frRahmen -borderwidth 5 -relief solid -width ↩
    100 -height 50
4 ttk::label .frRahmen.lbLabel -text "Hallo"
5 pack .frRahmen
6 pack propagate .frRahmen 0
7 pack .frRahmen.lbLabel
```



In Zeile 6 wird mit dem Befehl `pack propagate` verhindert, dass sich der Rahmen automatisch an die Größe seines Inhalts anpasst. Er behält die mit den Optionen `-width` und `-height` festgelegte Größe.

43 Menü

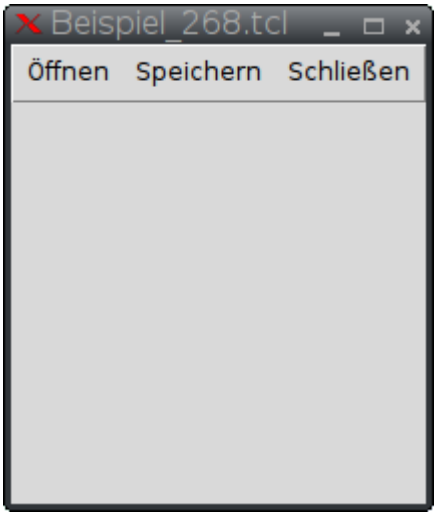
Man kann aufklappende Menüs, Pop-up-Menüs und Optionen-Menüs erstellen. Ein Menü kann neben den Texteinträgen folgende weiteren Elemente enthalten:

- Checkbutton
- Radiobutton
- Separator

43.1 Menü

Listing 43.1: Einfaches Menü (Beispiel268.tcl)

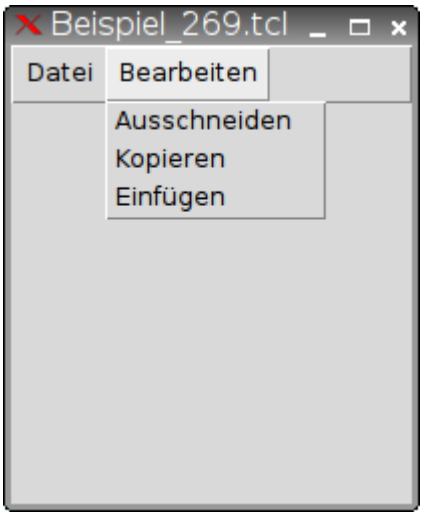
```
1 #!/usr/bin/env wish
2
3 option add *Menu.tearOff 0
4 . configure -menu .m
5 menu .m
6 .m add command -label "Oeffnen" -command tk_getOpenFile
7 .m add command -label "Speichern" -command tk_getSaveFile
8 .m add command -label "Schliessen" -command exit
```



In Zeile 3 wird verhindert, dass das Menü mit einer Trennlinie anfängt. Diese Zeile sollte man immer in den Programmcode einfügen. In Zeile 4 wird das Menü dem Hauptfenster zugeordnet. In Zeile 5 wird das Menü initialisiert. In den Zeilen 6 bis 8 werden Einträge dem Menü hinzugefügt.

Listing 43.2: Aufklappendes Menü (Beispiel269.tcl)

```
1  #!/usr/bin/env wish
2
3  option add *Menu.tearOff 0
4  . configure -menu .mbar
5
6  menu .mbar
7  .mbar add cascade -label "Datei" -menu .mbar.datei
8  .mbar add cascade -label "Bearbeiten" -menu ↩
     .mbar.bearbeiten
9
10 menu .mbar.datei
11 .mbar.datei add command -label "Oeffnen" -command ↩
     tk_getOpenFile
12 .mbar.datei add command -label "Speichern" -command ↩
     tk_getSaveFile
13 .mbar.datei add separator
14 .mbar.datei add command -label "Schliessen" -command exit
15
16 menu .mbar.bearbeiten
17 .mbar.bearbeiten add command -label "Ausschneiden"
18 .mbar.bearbeiten add command -label "Kopieren"
19 .mbar.bearbeiten add command -label "Einfuegen"
```



In Zeile 4 wird das Menü dem Hauptfenster (repräsentiert durch einen Punkt) zugeordnet. In Zeile 6 wird das Menü initialisiert. In den Zeilen 7 und 8 wird die oberste Menü-Ebene definiert. Beide Menüeinträge sind kaskadierend, d. h. es öffnet sich ein weiteres (Unter-) Menü. Die Zeilen 10 bis 14 definieren das Untermenü zum Menüeintrag `Datei`. Die Zeilen 16 bis 19 definieren das Untermenü zum Menüeintrag `Bearbeiten`.

Listing 43.3: Aufklappendes Menü mit Checkbuttons (Beispiel270.tcl)

```
1  #!/usr/bin/env wish
2
3  option add *Menu.tearOff 0
```

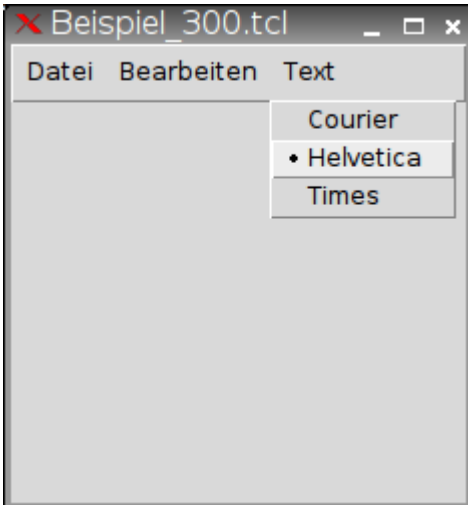
```
4  . configure -menu .mbar
5
6  menu .mbar
7  .mbar add cascade -label Datei -menu .mbar.datei
8  .mbar add cascade -label Bearbeiten -menu .mbar.bearbeiten
9  .mbar add cascade -label Text -menu .mbar.text
10
11 menu .mbar.datei
12 .mbar.datei add command -label "Oeffnen" -command ↩
     tk_getOpenFile
13 .mbar.datei add command -label "Speichern" -command ↩
     tk_getSaveFile
14 .mbar.datei add command -label "Schliessen" -command exit
15
16 menu .mbar.bearbeiten
17 .mbar.bearbeiten add command -label "Ausschneiden"
18 .mbar.bearbeiten add command -label "Kopieren"
19 .mbar.bearbeiten add command -label "Einfuegen"
20
21 menu .mbar.text
22 .mbar.text add checkbutton -label "Fett" -variable Fett
23 .mbar.text add checkbutton -label "Kursiv" -variable ↩
     Kursiv
24 .mbar.text add checkbutton -label "Unterstrichen" ↩
     -variable Unterstrichen
```

In den Zeilen 22 bis 24 werden Checkbuttons in das Menü eingefügt.

Listing 43.4: Aufklappendes Menü mit Radiobuttons (Beispiel300.tcl)

```
1  #!/usr/bin/env wish
2
3  option add *Menu.tearOff 0
4  . configure -menu .mbar
5
6  menu .mbar
```

```
7  .mbar add cascade -label Datei -menu .mbar.datei
8  .mbar add cascade -label Bearbeiten -menu .mbar.bearbeiten
9  .mbar add cascade -label Text -menu .mbar.text
10
11 menu .mbar.datei
12 .mbar.datei add command -label "Oeffnen" -command ↩
     tk_getOpenFile
13 .mbar.datei add command -label "Speichern" -command ↩
     tk_getSaveFile
14 .mbar.datei add command -label "Schliessen" -command exit
15
16 menu .mbar.bearbeiten
17 .mbar.bearbeiten add command -label "Ausschneiden"
18 .mbar.bearbeiten add command -label "Kopieren"
19 .mbar.bearbeiten add command -label "Einfuegen"
20
21 menu .mbar.text
22 .mbar.text add radiobutton -label "Courier" -variable ↩
     Schriftart
23 .mbar.text add radiobutton -label "Helvetica" -variable ↩
     Schriftart
24 .mbar.text add radiobutton -label "Times" -variable ↩
     Schriftart
25
26 set Schriftart "Helvetica"
```



In den Zeilen 22 bis 24 werden Radiobuttons in das Menü eingefügt. In Zeile 26 wird die anfängliche Schriftart festgelegt.
Man kann beim Anklicken eines Radiobuttons auch direkt einen Befehl ausführen:

Listing 43.5: Radiobuttons mit Befehlen verknüpfen (Beispiel324.tcl)

```
1 #!/usr/bin/env wish
2
3 proc ButtonKlick {Button} {
4   .lbButtonKlick configure -text $Button
```

```
5    update idletasks
6  }
7
8  wm minsize . 270 40
9
10 option add *Menu.tearOff 0
11 . configure -menu .m
12 menu .m
13 .m add command -label "Schliessen" -command exit
14 .m add cascade -label "Auswahl" -menu .m.auswahl
15
16 menu .m.auswahl
17 .m.auswahl add radiobutton -label "Button 1" -variable ↩
     Button -command {ButtonKlick $Button}
18 .m.auswahl add radiobutton -label "Button 2" -variable ↩
     Button -command {ButtonKlick $Button}
19
20 ttk::label .lbButton -text "Ihre Auswahl:"
21 ttk::label .lbButtonKlick -text ""
22
23 grid .lbButton -row 0 -column 0 -padx 3 -pady 3
24 grid .lbButtonKlick -row 0 -column 1 -padx 3 -pady 3
```

Wenn man das Programm startet und im Menü auf `Auswahl` Button 1| klickt, sieht man folgendes:

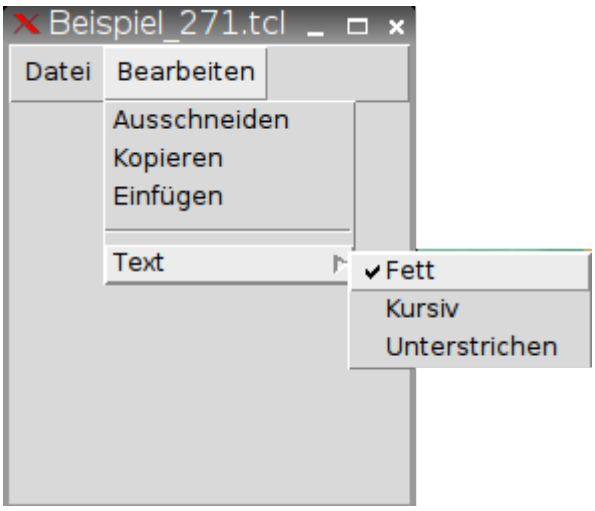
Wenn man im Menü auf `Auswahl` Button 2| klickt, ändert sich der Text wie folgt:

In den Zeilen 17 und 18 werden die Radiobuttons um einen `command`-Befehl ergänzt, der die Prozedur `ButtonKlick` aufruft. Die Zeilen 3 bis 6 umfassen die Prozedur `ButtonKlick`. In Zeile 4 wird der Text des GUI-Elements entsprechend dem angeklickten Button geändert. In Zeile 5 wird die GUI aktualisiert.

Listing 43.6: Kaskadierendes Menü (Beispiel271.tcl)

```
1 #!/usr/bin/env wish
2
3 option add *Menu.tearOff 0
4 . configure -menu .mbar
```

```
5
6  menu .mbar
7  .mbar add cascade -label Datei -menu .mbar.datei
8  .mbar add cascade -label Bearbeiten -menu .mbar.bearbeiten
9
10 menu .mbar.datei
11 .mbar.datei add command -label "Oeffnen" -command ↩
     tk_getOpenFile
12 .mbar.datei add command -label "Speichern" -command ↩
     tk_getSaveFile
13 .mbar.datei add separator
14 .mbar.datei add command -label "Schliessen" -command exit
15
16 menu .mbar.bearbeiten
17 .mbar.bearbeiten add command -label "Ausschneiden"
18 .mbar.bearbeiten add command -label "Kopieren"
19 .mbar.bearbeiten add command -label "Einfuegen"
20 .mbar.bearbeiten add separator
21 .mbar.bearbeiten add cascade -label "Text" -menu ↩
     .mbar.bearbeiten.text
22
23 menu .mbar.bearbeiten.text
24 .mbar.bearbeiten.text add checkbutton -label "Fett" ↩
     -variable Fett
25 .mbar.bearbeiten.text add checkbutton -label "Kursiv" ↩
     -variable Kursiv
26 .mbar.bearbeiten.text add checkbutton -label "↩
     Unterstrichen" -variable Unterstrichen
```



Vor allem bei mehrsprachigen Programmen ist es erforderlich, die Beschriftung des Menüs je nach gewählter Sprache zu ändern (siehe Kapitel 55 auf Seite 691).

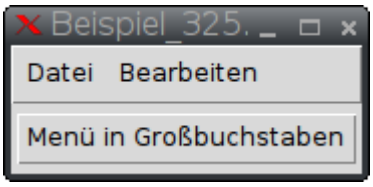
Listing 43.7: Beschriftung des Menüs nachträglich ändern (Beispiel325.tcl)

```
1 #!/usr/bin/env wish
2
```

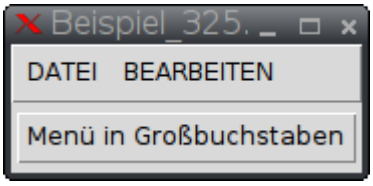
```
3  proc MenuGross {} {
4    .mbar entryconfigure 0 -label "DATEI"
5    .mbar entryconfigure 1 -label "BEARBEITEN"
6
7    .mbar.datei entryconfigure 0 -label "OEFFNEN"
8    .mbar.datei entryconfigure 1 -label "SPEICHERN"
9    .mbar.datei entryconfigure 3 -label "SCHLIESSEN"
10
11   .mbar.bearbeiten entryconfigure 0 -label "AUSSCHNEIDEN"
12   .mbar.bearbeiten entryconfigure 1 -label "KOPIEREN"
13   .mbar.bearbeiten entryconfigure 2 -label "EINFUEGEN"
14   .mbar.bearbeiten entryconfigure 4 -label "TEXT"
15
16   .mbar.bearbeiten.text entryconfigure 0 -label "FETT"
17   .mbar.bearbeiten.text entryconfigure 1 -label "KURSIV"
18   .mbar.bearbeiten.text entryconfigure 2 -label "↩
       UNTERSTRICHEN"
19
20   update idletasks
21 }
22
23 option add *Menu.tearOff 0
24 . configure -menu .mbar
25
26 menu .mbar
27 .mbar add cascade -label "Datei" -menu .mbar.datei
28 .mbar add cascade -label "Bearbeiten" -menu ↩
     .mbar.bearbeiten
29
30 menu .mbar.datei
31 .mbar.datei add command -label "Oeffnen" -command ↩
     tk_getOpenFile
32 .mbar.datei add command -label "Speichern" -command ↩
     tk_getSaveFile
33 .mbar.datei add separator
34 .mbar.datei add command -label "Schliessen" -command exit
35
36 menu .mbar.bearbeiten
37 .mbar.bearbeiten add command -label "Ausschneiden"
38 .mbar.bearbeiten add command -label "Kopieren"
39 .mbar.bearbeiten add command -label "Einfuegen"
40 .mbar.bearbeiten add separator
41 .mbar.bearbeiten add cascade -label "Text" -menu ↩
     .mbar.bearbeiten.text
42
43 menu .mbar.bearbeiten.text
44 .mbar.bearbeiten.text add checkbutton -label "Fett" ↩
     -variable Fett
45 .mbar.bearbeiten.text add checkbutton -label "Kursiv" ↩
     -variable Kursiv
46 .mbar.bearbeiten.text add checkbutton -label "↩
     Unterstrichen" -variable Unterstrichen
47
```

```
48 ttk::button .btGross -text "Menue in Grossbuchstaben" ↲
     -command MenuGross
49 grid .btGross -row 0 -column 0 -padx 3 -pady 5
```

Nach dem Starten des Programms besteht das Menü aus Groß- und Kleinbuchstaben.



Wenn man auf den Button `Menü in Großbuchstaben` klickt, wird das Menü geändert.



In den Zeilen 3 bis 21 sieht man die Prozedur `MenuGross`. In Zeile 4 wird die Beschriftung für das erste Element im Menü `.mbar` geändert. Das erste Element hat den Index `0`. Der Befehl hat folgenden Aufbau:
`Pfadname entryconfigure Index Option NeuerWert`
Der Pfadname ist `.mbar`, der Index ist `0`, die Option heißt `-label` und der neue Wert ist `"DATEI"`. Beim Index muss man beachten, dass auch zum Beispiel Trennlinien mitgezählt werden. Deshalb ist in Zeile 9 der Index für den Menüeintrag `Schließen` nicht `2`, sondern `3`. In Zeile 20 wird die Bildschirmanzeige aktualisiert.

43.2 Popup-Menü

Befehle:

- `tk_popup MenüName %X %Y`
- `MenüName entryconfigure Index -Option Wert`

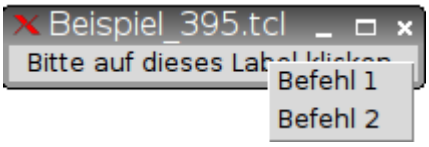
Mit dem Befehl `tk_popup` kann man ein Menü anzeigen, das an ein Element (z. B. eine Listbox) gekoppelt ist. Wahrscheinlich kennen Sie ein solches Kontextmenü vom Dateimanager. Wenn man mit der rechten Maustaste auf eine Datei klickt, erscheint ein Menü mit den Befehlen `kopieren`, `ausschneiden`, `einfügen` usw. Der Befehl `tk_popup` erwartet als erstes Argument den Namen des Menüs. Danach legt man die Position im Fenster fest, an der das Menü erscheine soll. Üblicherweise ist das die x/y-Koordinate des Elements, zu dem das Menü gehört. Wenn man später Einträge im Menü ändern möchte (z. B. soll der Eintrag `Einfügen` erst aktiv sein, wenn zuvor `Kopieren` angeklickt wurde), verwendet man den Befehl `entryconfigure`.

Listing 43.8: Popup-Menü erstellen (Beispiel395.tcl)

```
1 #!/usr/bin/env wish
```

```
2
3 option add *Menu.tearOff 0
4 menu .popupMenu
5 .popupMenu add command -label "Befehl 1" -command bell
6 .popupMenu add command -label "Befehl 2" -command bell
7
8 label .lb -text "Bitte auf dieses Label klicken."
9 pack .lb
10 bind .lb <ButtonPress-3> {tk_popup .popupMenu %X %Y}
```

Klicken Sie mit der rechten Maustaste auf das Textfeld:

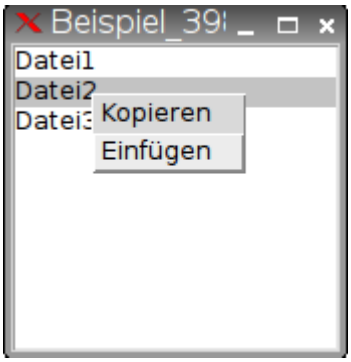


In den Zeilen 4 bis 6 wird ein Menü mit zwei Einträgen erzeugt. In Zeile 10 wird das Menü als Popup-Menü dem Label-Element zugeordnet. Der `bind`-Befehl wird verwendet, um ein Tastatur-Ereignis oder einen Mausklick mit einer Aktion zu verknüpfen (siehe Kapitel 44 auf Seite 533). In diesem Fall wird als Ereignis `<ButtonPress-3>` festgelegt. Dies ist ein Klick mit der rechten Maustaste.

Listing 43.9: Popup-Menü ändern (Beispiel398.tcl)

```
#!/usr/bin/env wish

set Liste {Datei1 Datei2 Datei3}

option add *Menu.tearOff 0
menu .popupDateien
.popupDateien add command -label "Kopieren" -state normal ↩
   -command {.popupDateien entryconfigure 1 -state normal}
.popupDateien add command -label "Einfuegen" -state ↩
   disabled

listbox .lb -listvariable Liste
pack .lb
bind .lb <ButtonPress-3> {tk_popup .popupDateien %X %Y}
```

Nachdem man im Popup-Menü auf `Kopieren` geklickt hat, wird der Eintrag `Einfügen` freigeschaltet:



In Zeile 8 ist der Status des Menüeintrags `Einfügen` zunächst deaktiviert (`-state disabled`). In Zeile 7 wird mit der `command`-Option festgelegt, dass der Menüeintrag `Einfügen` (dieser hat den Index `1`) aktiviert werden soll (`-state normal`).

43.3 Optionen-Menü

Befehl:

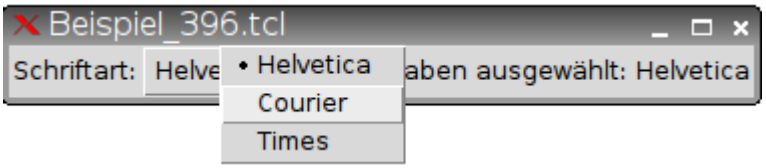
- `tk_optionMenu MenüName Variable Option1 Option2 ...`

Mit dem Befehl `tk_optionMenu` kann man ein Menü erzeugen, das dem Anwender verschiedene Optionen zur Auswahl gibt. Wenn der Anwender eine Option anklickt, wird diese Option in einer Variablen gespeichert.

Listing 43.10: Optionen-Menü (Beispiel396.tcl)

```
1  #!/usr/bin/env wish
2
3  label .lb1 -text "Schriftart:"
4  tk_optionMenu .optmenu Schriftart Helvetica Courier Times
5  label .lb2 -text "Sie haben ausgewaehlt:"
6  label .lb3 -textvariable Schriftart
7
8  pack .lb1 -side left
9  pack .optmenu -side left
10 pack .lb2 -side left
11 pack .lb3 -side left
```

Klicken Sie mit der Maus auf den Button mit der Schriftart. Es erscheint ein Menü mit den Schriftarten.



In Zeile 2 wird das Optionen-Menü definiert. Als erstes Argument wird der Name des Menüs festgelegt, danach folgt der Variablenname. Und daran anschließend folgen die einzelnen Optionen.

44 Maus- und Tastaturereignisse (Bindings)

44.1 Binding für ein Element

Befehl:

- `bind Elementname <modifier-modifier ... -modifier-type-detail> {Anweisung}`

Der `bind`-Befehl verknüpft die Elemente mit Mausereignissen (z. B. linke Maustaste drücken) und Tastaturereignissen (z. B. Return-Taste drücken) und legt fest, welche Anweisung ausgeführt werden soll, wenn das Ereignis eintritt. Der Befehl besteht aus folgenden Parametern:

`modifier` = das sind die Tasten `Strg`, `Alt` usw. oder die Maustasten.
`type` = das ist das Ereignis (z. B. `Keypress`, `ButtonPress`)
`detail` = es definiert die Taste (z.B. linke Maustaste oder Buchstabe `q` auf der Tastatur)
`Anweisung` = beinhaltet alle Befehle die ausgeführt werden sollen, wenn das Ereignis stattfindet. Meistens wird das der Aufruf einer Prozedur sein.

Die `modifier`- und die `detail`-Angabe sind optional. Wenn man z. B. beim `KeyPress`-Ereignis keinen Buchstaben festlegt, wird der Befehl bei jedem Tastendruck (egal welcher Buchstabe) ausgeführt. Man kann mehrere Modifier miteinander kombinieren (z. B. `Button1-Double` oder `Shift-Alt`).

Tabelle 44.1: Die wichtigsten Modifier

Modifier	Beschreibung
`Control`	Strg-Taste
`Shift`	Umschalttaste Großschreibung
`Lock`	Umschalttaste Groß-/Kleinschreibung
`Alt`	Alt-Taste
`Button1`	linke Maustaste
`Button2`	mittlere Maustaste
`Button3`	rechte Maustaste
`Double`	Doppelklick mit der Maus

Tabelle 44.2: Die wichtigsten Ereignisse (type)

Type	Beschreibung
`KeyPress`	Eine Taste wird gedrückt
`KeyRelease`	Eine Taste wird losgelassen
`ButtonPress`	Ein Mausbutton wird gedrückt
`ButtonRelease`	Ein Mausbutton wird losgelassen
`Enter`	Der Mauszeiger wird in ein Element hineinbewegt
`Leave`	Der Mauszeiger wird aus einem Element herausbewegt
`Motion`	Der Mauszeiger wird innerhalb eines Elements bewegt
`MouseWheel`	Das Mausrad wird gedreht
`FocusIn`	Ein Element bekommt den Tastaturfokus
`FocusOut`	Ein Element verliert den Tastaturfokus
`Configure`	Ein Element wird angezeigt oder geändert
`Destroy`	Ein Element wird gelöscht

Das `KeyPress`-Ereignis ist nur bei Elementen möglich, die einen Eingabefokus bekommen können (z. B. ein Entry-Element). Man kann deshalb das Ereignis z. B. nicht mit einem Canvas-Element kombinieren.
Den bind-Befehl bildet man am besten in folgender Reihenfolge:

1.) den `type` festlegen (z.B. `KeyPress`, `ButtonPress`, `Motion`)

2.) wenn es sich um den `type` `KeyPress`, `KeyRelease`, `ButtonPress` oder `ButtonRelease` handelt, dann das `detail` festlegen (Buchstabe oder Buttonnummer). Zum Beispiel: `<ButtonPress-1>` für die linke Maustaste.

3.) Wenn gewünscht, einen oder mehrere `modifier` (z.B. `Alt`- oder `Strg`-Taste oder einen Mausbutton) hinzufügen.

Auch das Drehen des Mausrads kann als Ereignis registriert werden:

Tabelle 44.3: Mausrad

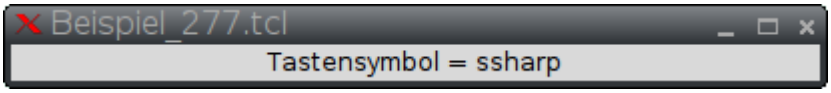
Befehl	Beschreibung
`<Button-4>`	Nach oben scrollen
`<Button-5>`	Nach unten scrollen

Einige Elemente (z. B. Treeview) verfügen standardmäßig über Tastatur- oder Mausereignisse. Wenn man ein solches Ereignis abschalten möchte, kann man das Ereignis mit dem Befehl `break` verknüpfen (siehe späteres Beispiel H).
Um das Tastensymbol für die `detail`-Angabe herauszufinden, kann man folgendes Miniprogramm verwenden:

Listing 44.1: Tastensymbol ermitteln (Beispiel277.tcl)

```
#!/usr/bin/env wish

ttk::label .lb -text "Taste druecken"
pack .lb
bind .lb <Key> {.lb configure -text "Tastensymbol = %K"}
focus .lb
```

Nachdem die ß-Taste gedrückt wurde:



Nachfolgende einige Beispiele:

Beispiel A: Eingabefeld mit der `Return`-Taste verbinden
Windows und Linux haben unterschiedliche Ereignisse, deshalb sollte man immer beide nachstehenden Ereignisse setzen.

```
bind .entry <KeyPress-Return> {Anweisung}
bind .entry <KP_Enter> {Anweisung}
```

Beispiel B: Eingabefeld mit der `Escape`-Taste verbinden

```
bind .entry <KeyPress-Escape> {Anweisung}
```

Beispiel C: Eingabefeld mit der Tastenkombination `Strg+q` verbinden

```
bind .entry <Control-KeyPress-q> {Anweisung}
```

Beispiel D: Mauszeiger wird in ein Eingabefeld bewegt

```
bind .entry <Enter> {Anweisung}
```

Beispiel E: Mausklick mit der linken Taste während die `Shift`-Taste gedrückt ist

```
bind .entry <Shift-ButtonPress-1> {Anweisung}
```

Beispiel F: Doppelter Mausklick mit der rechten Taste
```
bind .entry <Double-ButtonPress-3> {Anweisung}
```

Beispiel G: Mausrad drehen
```
bind .table <Button-4> {Anweisung}
bind .table <Button-5> {Anweisung}
```

Beispiel H: Ereignis abschalten (z. B. bei dem Treeview-Element `.tv` das Ereignis `Shift+Mausklick` abschalten)
```
bind .tv <Shift-ButtonPress> break
```

44.2 Binding für das gesamte Programm

Befehl:

- `bind all <modifier-modifier ...`
  `-modifier-type-detail> {Anweisung}`

Mit dem Befehl `bind all` kann man Ereignisse festlegen, die für das gesamte Programm gelten. Beispiel:
```
bind all <Motion> {puts "Mauszeiger ist an Position %x, %y}
```

44.3 Substitution bei Bindings

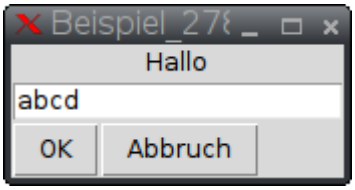
Bei der Ausführung der Anweisung zu einem Ereignis ist es oft erforderlich, das auslösende Elemente oder dessen Eigenschaft zu kennen. Dazu gibt es die Substitutionen.

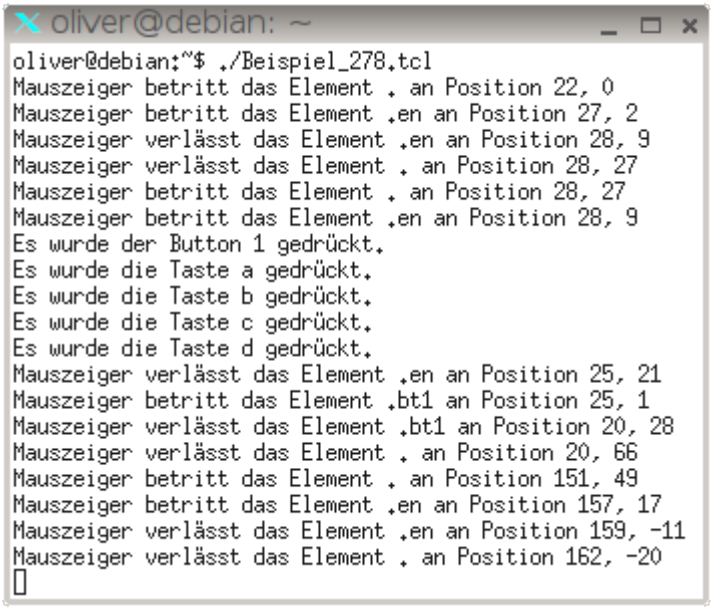
Tabelle 44.4: Die wichtigsten Substitutionen

Substitution	Beschreibung
`%W`	Name des Elements
`%b`	Mausbutton
`%x`	x-Koordinate des Mauszeigers
`%y`	y-Koordinate des Mauszeigers
`%A`	Unicode-Zeichen der Taste
`%K`	Tastatursymbol
`%X`	x-Koordinate des Mauszeigers relativ zum Hauptfenster
`%Y`	y-Koordinate des Mauszeigers relativ zum Hauptfenster
`%h`	Höhe des Elements
`%w`	Breite des Elements

Listing 44.2: Substitution (Beispiel278.tcl)

```
#!/usr/bin/env wish

ttk::label .lb -text "Hallo"
ttk::entry .en
ttk::button .bt1 -text "OK"
ttk::button .bt2 -text "Abbruch"

pack .lb -side top
pack .en -side top
pack .bt1 -side left
pack .bt2 -side left

bind all <Motion> {puts "Mauszeiger ist an Position %x, %y"}
bind all <Enter> {puts "Mauszeiger betritt das Element %W an Position %x, %y"}
bind all <Leave> {puts "Mauszeiger verlaesst das Element %W an Position %x, %y"}
bind .en <ButtonPress> {puts "Es wurde der Button %b gedrueckt."}
bind .en <KeyPress> {puts "Es wurde die Taste %A gedrueckt."}
```





44.4 Virtuelle Ereignisse

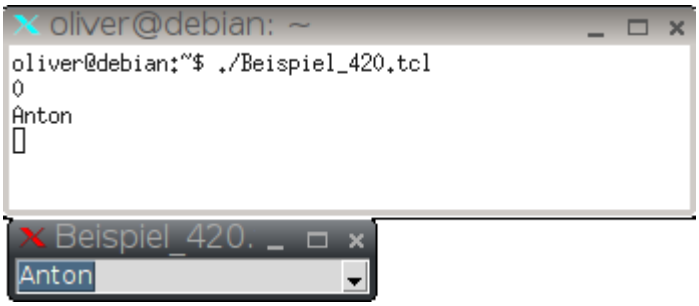
Einige Elemente verfügen über virtuelle Ereignisse, dies ausgelöst werden, wenn der Anwender z. B. einen Eintrag auswählt.

Tabelle 44.5: Elemente und ihre virtuelle Ereignisse

Element	Virtuelles Ereignis
`listbox`	`<<ListboxSelect>>`
`ttk::combobox`	`<<ComboboxSelected>>`
`ttk::notebook`	`<<NotebookTabChanged>>`

Listing 44.3: Virtuelles Ereignis der Combobox (Beispiel420.tcl)

```
#!/usr/bin/env wish

set Liste {Anton Berta Caesar Dora}
ttk::combobox .cb -values $Liste -textvariable Auswahl ↲
   -state readonly
pack .cb
bind .cb <<ComboboxSelected>> {puts [%W current]; puts ↲
   $Auswahl}
```



Das virtuelle Ereignis `<<ComboboxSelected>>` wird ausgelöst, wenn der Anwender einen Eintrag aus der Liste wählt. `[%W current]` gibt den Index des ausgewählten Eintrags an.

Listing 44.4: Virtuelles Ereignis der Listbox (Beispiel421.tcl)

```
#!/usr/bin/env wish

set Liste {Anton Berta Caesar Dora}
listbox .lb -listvariable Liste -selectmode single
pack .lb
bind .lb <<ListboxSelect>> {puts [%W curselection]}
```

Wenn der Anwender einen Eintrag auswählt, wird das virtuelle Ereignis `<<ListboxSelect>>` ausgelöst. `[%W curselection]` gibt den Index des ausgewählten Eintrags an.

Listing 44.5: Virtuelles Ereignis des Notebooks (Beispiel422.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::notebook .n
4  ttk::frame .n.f1
5  ttk::frame .n.f2
6  .n add .n.f1 -text "Reiter 1"
7  .n add .n.f2 -text "Reiter 2"
8
9  ttk::label .n.f1.l1 -text "Seite 1"
10 ttk::button .n.f1.b1 -text "OK"
11 ttk::label .n.f2.l1 -text "Seite 2"
12
13 pack .n -fill both -expand 1
14 pack .n.f1.l1 -side top
15 pack .n.f1.b1 -side bottom
16 pack .n.f2.l1 -side top
17
18 bind .n <<NotebookTabChanged>> {puts [%W index current]}
19 .n select .n.f1
```

In Zeile 18 wird das virtuelle Ereignis `<<NotebookTabChanged>>` definiert. Es wird aus-

gelöst, wenn eine Reiter des Notebbooks angeklickt (bzw. selektiert) wird. `[%W index current]` gibt den Index des angeklickten Reiters an. In Zeile 19 wird der erste Reiter (hat den Index 0) vom Programm selektiert. Dies führt dazu, dass das virtuelle Ereignis ausgeführt wird.

44.5 Alle Tastaturereignisse ignorieren

Befehl:

- `bind Elementname <KeyPress> break`

Mit dem Befehl kann man z. B. während einer lange dauernden Prozedur alle Tastatureingaben verhindern. Als Elementname kann entweder das Hauptfenster (repäsentiert durch einen Punkt) oder ein bestimmtes Element angegeben werden. Häufig wird für die Zeitdauer der blockierten Tastatur auch das Symbol des Mauszeigers geändert (z. B. in eine Sanduhr).

44.6 Mausklicks eine Zeit lang ignorieren

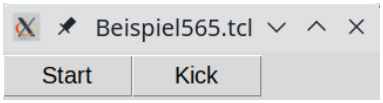
Befehl:

- `tk busy`

Wenn das Programm eine Prozedur ausführt, die eine längere Zeit läuft, kann es passieren, dass der Benutzer in der Zwischenzeit einen Button anklickt. Dieses Anklicken wird vom Programm registriert und nach der Prozedur ausgeführt. Zum Ausprobieren, kann man das folgende Beispiel verwenden: wenn man den `Start`-Button anklickt, wird eine Prozedur aufgerufen, die drei Sekunden läuft. Klickt man in der Zwischenzeit mehrmals auf den `Klick`-Button, wird das Ereignis registriert und nach der `Start`-Pozedur ausgeführt.

Listing 44.6: Mausklicks werden ausgeführt (Beispiel565.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Start {} {
4    puts "Start"
5    for {set i 1} {$i <= 3} {incr i} {
6      after 1000
7    }
8    puts "Ende"
9  }
10
11 proc Klick {} {
12   puts "Klick"
13 }
14
15 ttk::button .btStart -text "Start" -command {Start}
16 ttk::button .btKlick -text "Kick" -command {Klick}
17 pack .btStart -side left
18 pack .btKlick -side left
```



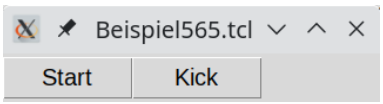

```
oliver@pc:~$ ./Beispiel565.tcl
Start
Ende
Klick
Klick
Klick
Klick
Klick
oliver@pc:~$
```

Falls man aber während der Prozedur-Ausführung verhindern will, dass der Benutzer einen Button anklickt, muss man den Befehl `tk busy` in Verbindung mit `update` hinzufügen.

Listing 44.7: Mausklicks werden ignoriert (Beispiel566.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Start {} {
4    puts "Start"
5    tk busy hold .
6    update
7    for {set i 1} {$i <= 3} {incr i} {
8      after 1000
9    }
10   tk busy forget .
11   puts "Ende"
12 }
13
14 proc Klick {} {
15   puts "Klick"
16 }
17
18 ttk::button .btStart -text "Start" -command {Start}
19 ttk::button .btKlick -text "Kick" -command {Klick}
20 pack .btStart -side left
21 pack .btKlick -side left
```

In den Zeilen 5 und 6 wird das gesamte Programmfenster gesperrt, so dass während der Prozedur keine Mausklicks registriert werden. Am Ende der Prozedur wird in Zeile 10 die Sperre wieder aufgehoben. Wie man sieht, werden keine Mausklicks, die man während der Prozedur-Ausführung gemacht hat, nach der Prozedur ausgeführt.



```
oliver@pc:~$ ./Beispiel566.tcl
Start
Ende
oliver@pc:~$
```

Eine ausführliche Beschreibung des `tk busy`-Befehls findet man im Kapitel 48 auf Seite 551 .

45 Mauszeiger

Befehle:

- `set CursorAlt [Fenstername cget -cursor]`
- `Fenstername configure -cursor watch`
- `Fenstername configure -cursor $CursorAlt`

Tabelle 45.1: Mauszeiger

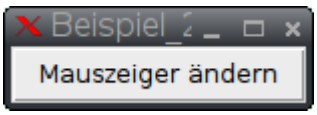
A-D	E-R	S-Z
arrow	exchange	sailboat
based_arrow_down	fleur	sb_down_arrow
based_arrow_up	gobbler	sb_h_double_arrow
boat	gumby	sb_left_arrow
bogosity	hand1	sb_right_arrow
bottom_left_corner	hand2	sb_up_arrow
bottom_right_corner	heart	sb_v_double_arrow
bottom_side	icon	shuttle
bottom_tee	iron_cross	sizing
box_spiral	left_ptr	spider
center_ptr	left_side	spraycan
circle	left_tee	star
clock	leftbutton	target
coffee_mug	ll_angle	tcross
cross	lr_angle	top_left_arrow
cross_reverse	man	top_left_corner
crosshair	middlebutton	top_right_corner
diamond_cross	mouse	top_side
dot	none	top_tee

Tabelle 45.1: Mauszeiger

A-D	E-R	S-Z
dotbox	pencil	trek
double_arrow	pirate	ul_angle
draft_large	plus	umbrella
draft_small	question_arrow	ur_angle
draped_box	right_ptr	watch
	right_side	xterm
	right_tee	X_cursor
	rightbutton	
	rtl_logo	

Listing 45.1: Mauszeiger ändern (Beispiel279.tcl)

```
1  #!/usr/bin/env wish
2
3  proc MauszeigerAendern {} {
4    set CursorAlt [. cget -cursor]
5    . configure -cursor watch
6    update
7    after 3000
8    . configure -cursor $CursorAlt
9    update
10 }
11
12 ttk::button .btStart -text "Mauszeiger aendern" -command ↩
     MauszeigerAendern
13 pack .btStart
```



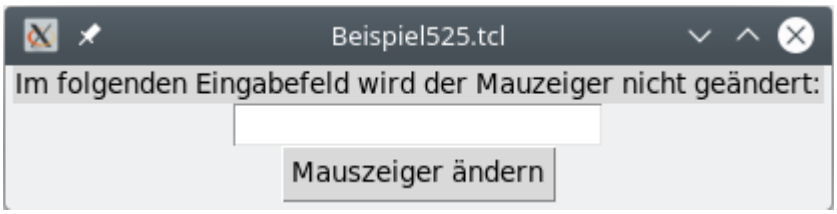
In der Zeile 4 wird das Symbol des Mauszeigers gespeichert. In Zeile 5 wird das Symbol des Mauszeigers geändert. In Zeile 6 wird die grafische Darstellung aktualisiert. In Zeile 7 wartet das Programm drei Sekunden. In Zeile 8 wird das ursprüngliche Symbol des Mauszeigers wieder hergestellt. In Zeile 9 wird die grafische Darstellung aktualisiert.

Wenn man den Mauszeiger ändert, muss man beachten, dass es Elemente gibt, die eine eigene Einstellung für den Mauszeiger haben (z. B. das `entry`-Element). Somit ist es ggf. nicht ausreichend, den Mauszeiger nur für das Hauptfenster zu ändern, sondern man muss den Mauszeiger eventuell auch für einzelne Elemente umstellen. Deshalb ist es

meistens besser, den Befehl `tk busy` (siehe Kapitel 48 auf Seite 551) zu verwenden, wenn man den Mauszeiger zeitweilig für das gesamte Programm ändern will.

Listing 45.2: Mauszeiger ändert sich nicht bei jedem Element (Beispiel525.tcl)

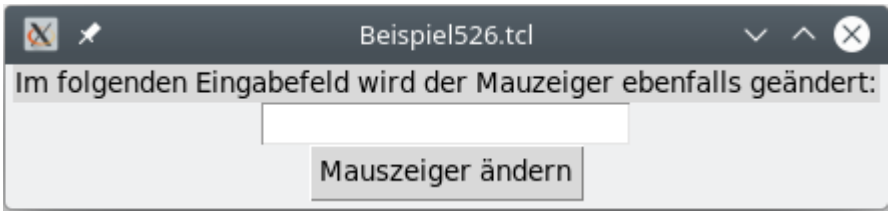
```
#!/usr/bin/env wish

proc MauszeigerAendern {} {
  set CursorAlt [. cget -cursor]
  . configure -cursor watch
  update
}

ttk::label .lbText -text "Im folgenden Eingabefeld wird der Mauzeiger nicht geaendert:"
ttk::entry .enText
ttk::button .btStart -text "Mauszeiger aendern" -command MauszeigerAendern
pack .lbText
pack .enText
pack .btStart -side bottom
```



Bei einem Klick auf den Button wird der Mauszeiger für das Hauptfenster geändert. Wenn man allerdings den Mauzeiger über das Eingabefeld zieht, nimmt der Mauszeiger die ursprüngliche Form an.

Listing 45.3: Mauszeiger ändert sich auch beim Entry-Element (Beispiel526.tcl)

```
#!/usr/bin/env wish

proc MauszeigerAendern {} {
  set CursorAlt [. cget -cursor]
  . configure -cursor watch
  .enText configure -cursor watch
  update
}

ttk::label .lbText -text "Im folgenden Eingabefeld wird der Mauzeiger ebenfalls geaendert:"
ttk::entry .enText
ttk::button .btStart -text "Mauszeiger aendern" -command MauszeigerAendern
pack .lbText
pack .enText
pack .btStart -side bottom
```



Durch den Befehl in Zeile 6 wird der Mauszeiger auch für das `entry`-Element geändert.

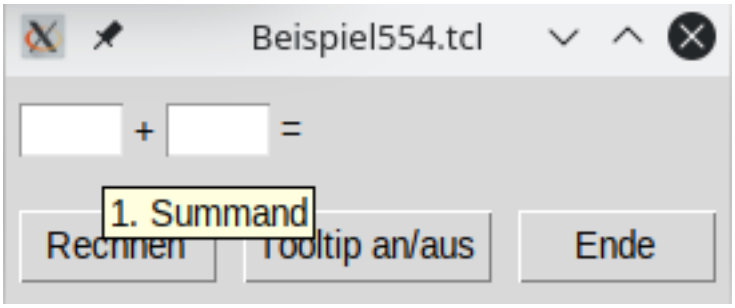
46 Tooltip anzeigen

Man kann die einzelnen Elemente mit einem Tooltip ausstatten. Das bedeutet, dass ein kurzer Hinweistext angezeigt wird, wenn sich der Mauszeiger über dem Element befindet.

Listing 46.1: Tooltip anzeigen (Beispiel554.tcl)

```
#!/usr/bin/env wish

package require tooltip

set TooltipStatus "an"

proc TooltipAnAus {TooltipStatus} {
  if {$TooltipStatus eq "an"} {
    ::tooltip::tooltip off
    set TooltipStatus "aus"
  } else {
    ::tooltip::tooltip on
    set TooltipStatus "an"
  }
  return $TooltipStatus
}

ttk::frame .frAlles
ttk::frame .frAlles.frAufgabe
ttk::frame .frAlles.frButton

ttk::entry .frAlles.frAufgabe.enZahl1 -textvariable Zahl1 -width 5
ttk::entry .frAlles.frAufgabe.enZahl2 -textvariable Zahl2 -width 5
ttk::label .frAlles.frAufgabe.lbErgebnis -textvariable Ergebnis -width 5

ttk::label .frAlles.frAufgabe.lbPlus -text "+"
ttk::label .frAlles.frAufgabe.lbGleich -text "="

ttk::button .frAlles.frButton.btRechnen -text "Rechnen" -command {set Ergebnis [Rechnen $Zahl1 $Zahl2]}
ttk::button .frAlles.frButton.btTooltipAnAus -text "Tooltip an/aus" -command {set TooltipStatus [TooltipAnAus $TooltipStatus]}
ttk::button .frAlles.frButton.btEnde -text "Ende" -command exit

pack .frAlles.frAufgabe -side top -fill x -padx 5 -pady 10
pack .frAlles.frAufgabe.enZahl1 -side left
pack .frAlles.frAufgabe.lbPlus -side left -padx 3
```

```
36 pack .frAlles.frAufgabe.enZahl2 -side left
37 pack .frAlles.frAufgabe.lbGleich -side left -padx 3
38 pack .frAlles.frAufgabe.lbErgebnis -side left
39
40 pack .frAlles.frButton -side top -fill x -pady 10
41 pack .frAlles.frButton.btRechnen -side left -padx 5
42 pack .frAlles.frButton.btTooltipAnAus -side left -padx 5
43 pack .frAlles.frButton.btEnde -side left -padx 5
44
45 pack .frAlles -side top -expand yes -fill both
46 focus .frAlles.frAufgabe.enZahl1
47
48 ::tooltip::tooltip .frAlles.frAufgabe.enZahl1 "1. Summand"
49 ::tooltip::tooltip .frAlles.frAufgabe.enZahl2 "2. Summand"
50 ::tooltip::tooltip .frAlles.frButton.btRechnen "Startet ↩
     die Berechnung"
51 ::tooltip::tooltip .frAlles.frButton.btEnde "Beendet das ↩
     Programm"
```



Der Mauszeiger befindet sich über dem ersten Eingabefeld, so dass der gelb unterlegte Hinweis erscheint. In Zeile 3 wird das Paket `tooltip` eingebunden. In Zeile 5 wird die Variable `TooltipStatus` definiert, die speichert, ob Tooltips aktiv oder inaktiv sind. In den Zeilen 7 bis 16 werden die Tooltips an- oder ausgeschaltet. In den Zeilen 48 bis 51 werden die Tooltip-Texte für die Elemente festgelegt. Wenn der Tooltip-Text zu lang ist, kann man mit `\n` einen Zeilenumbruch im Tooltip-Text einfügen.

47 Button programmseitig ausführen

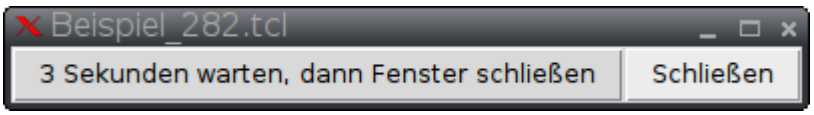
Befehle:

- `Buttonname flash` (nur beim klassischen Element)
- `Buttonname invoke`

Es kann vorkommen, dass man während der Ausführung des Programms einen Button ausführen (anklicken) möchte, ohne dass dies der Benutzer machen soll. Der Befehl `flash` sorgt für das optische Anklicken, der Button führt aber noch keinen Befehl aus. Den Befehl gibt es nur beim klassischen Button-Element, nicht beim ttk-Element. Der Befehl `invoke` führt den Button aus, genauso als hätte der Anwender den Button angeklickt.

Listing 47.1: Button programmseitig ausführen (Beispiel282.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Countdown {} {
4    after 3000
5    #.btSchliessen flash ; # gibt es nur beim klassischen ↲
       Element, nicht beim ttk-Element
6    .btSchliessen invoke
7  }
8  ttk::button .btStart -text "3 Sekunden warten, dann ↲
     Fenster schliessen" -command Countdown
9  ttk::button .btSchliessen -text "Schliessen" -command exit
10
11 pack .btStart -side left
12 pack .btSchliessen -side left
```



Wenn man auf den Button `3 Sekunden warten...` klickt, wird nach einer Pause von 3 Sekunden (Zeile 4) der Button `Schließen` ausgeführt (Zeilen 5 und 6).

48 Fenster oder Elemente vorübergehend sperren

Befehl:

- `tk busy`

Während einer lange dauernden Prozedur oder Berechnung möchte man verhindern, dass der Anwender weitere Elemente anklickt oder verändert. Dazu kann man das gesamte Fenster oder einzelne Elemente mit dem Befehl `tk busy` in Verbindung mit `update` vorübergehend sperren, das Programm also in einen Beschäftigt-Modus setzen.

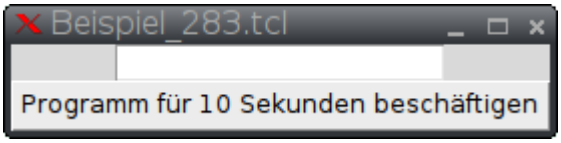
Tabelle 48.1: Die wichtigsten Befehle

Befehl	Beschreibung
`tk busy hold Elementname`	Sperrt ein Element
`tk busy configure Elementname -cursor watch`	Ändert den Mauszeiger
`tk busy forget Elementname`	Hebt die Sperre wieder auf

Listing 48.1: Programm in Beschäftigt-Modus setzen (Beispiel283.tcl)

```
#!/usr/bin/env wish

proc UmfangreicheProzedur {} {
  tk busy hold .
  tk busy configure . -cursor watch
  update

  after 10000

  tk busy forget .
  update
}

ttk::entry .en
ttk::button .bt -text "Programm fuer 10 Sekunden beschaeftigen" -command UmfangreicheProzedur
pack .en
pack .bt
```

Wenn man auf den Button klickt, wird die gesamte grafische Oberfläche für 10 Sekunden gesperrt.



In Zeile 4 wird das gesamte Fenster gesperrt. In Zeile 5 wird der Cursor während der Sperre in eine Uhr geändert. In Zeile 6 wird die Bildschirmausgabe aktualisiert und damit der `tk␣busy`-Befehl aktiviert. In Zeile 8 wird 10 Sekunden gewartet. Das steht exemplarisch für eine längere Berechnung oder Prozedur. In Zeile 10 wird das Fenster wieder freigegeben. In Zeile 11 wird die Bildschirmausgabe aktualisiert.

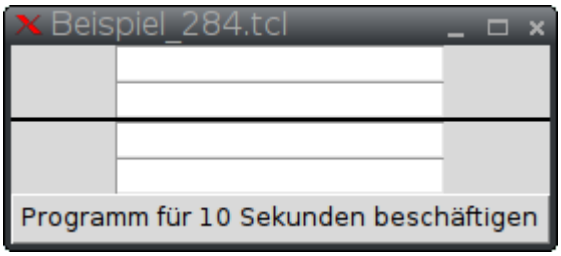
Man kann auch gezielt einzelne Elemente sperren. Allerdings wird man diese Möglichkeit nur selten brauchen, weil das Programm beschäftigt ist und während dessen keine weiteren Aktionen des Anwenders zulässt. Somit kann man auch gleich die ganze Anwendung sperren (siehe vorheriges Beispiel).

Listing 48.2: Einzelne Elemente sperren (Beispiel284.tcl)

```
1  #!/usr/bin/env wish
2
3  proc UmfangreicheProzedur {} {
4    focus .bt
5    tk busy hold .fr1
6    tk busy configure .fr1 -cursor watch
7    update
8
9    after 10000
10
11   tk busy forget .fr1
12   update
13 }
14
15 ttk::frame .fr1
16 ttk::entry .fr1.en1
17 ttk::entry .fr1.en2
18
19 ttk::frame .fr2
20 ttk::entry .fr2.en1
21 ttk::entry .fr2.en2
22
23 ttk::frame .fr3 -width 1 -height 2 -borderwidth 1 -relief ↩
     solid
24
25 ttk::button .bt -text "Programm fuer 10 Sekunden ↩
     beschaeftigen" -command UmfangreicheProzedur
26
27 pack .fr1.en1
28 pack .fr1.en2
29
30 pack .fr2.en1
31 pack .fr2.en2
32
33 pack .fr1
```

```
34 pack .fr3 -expand yes -fill x
35 pack .fr2
36 pack .bt
```

Klicken Sie auf den Button und bewegen Sie dann den Mauszeiger hoch und runter über die Eingabefelder.



In Zeile 4 wird sichergestellt, dass der Fokus nicht auf einem zu sperrenden Element ist. Dazu verlegt man den Fokus auf ein anderes, nicht zu sperrendes Element. In Zeile 5 wird der Frame mit den Eingabefeldern vorübergehend gesperrt. In Zeile 6 wird der Mauszeiger für dieses Element geändert. In Zeile 11 wird der Frame wieder freigegeben.

49 Mehrere Fenster öffnen

49.1 Toplevel-Dialog

Befehl:

- `toplevel`

Bisher wurde immer nur ein Fenster erzeugt, in dem die verschiedenen Elemente angezeigt wurden. Es gibt jedoch Situationen, in denen ein weiteres Fenster erscheinen soll. Zum Beispiel kann man abfragen, ob das Programm wirklich beendet werden soll oder welche Datei geöffnet werden soll. Bislang kennen Sie die vorgefertigten Dialoge zum Öffnen einer Datei (`tk_getOpenFile`), zum Auswählen eines Verzeichnisse (`tk_chooseDirectory`) , für die Ausgabe von Hinweisen (`tk_messageBox`) oder für einfache Rückfragen (`tk_dialog`). Mit dem Befehl `toplevel` kann man beliebige weitere Fenster erzeugen, die unabhängig vom ersten Fenster angezeigt und bedient werden können.

Listing 49.1: Zwei Fenster (Beispiel285.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Toplevel {} {
4    toplevel .top2
5    ttk::label .top2.lb -text "Toplevel"
6    pack .top2.lb
7  }
8
9  ttk::button .bt -text "Toplevel-Dialog erstellen" -command Toplevel
10 pack .bt
```

In Zeile 9 wird im ersten Fenster ein Button erzeugt. Wenn man auf den Button klickt, wird die Prozedur `Toplevel` aufgerufen, die ein weiteres Fenster erzeugt. In Zeile 4 wird ein zweites Fenster erstellt. In Zeile 5 wird ein Label in dem zweiten Fenster erzeugt. In Zeile 6 wird das Label angezeigt.

Listing 49.2: Beide Fenster sind aktiv (Beispiel286.tcl)

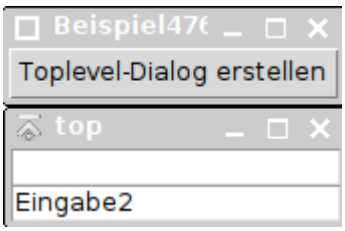
```
#!/usr/bin/env wish

proc Toplevel {} {
  toplevel .top2
  ttk::label .top2.lb -text "Toplevel"
  ttk::entry .top2.en
  pack .top2.lb
  pack .top2.en
}

ttk::entry .en
ttk::button .bt -text "Toplevel-Dialog erstellen" -command Toplevel
pack .en
pack .bt
```

Nachdem man auf den Button geklickt hat, erscheint ein zweites Fenster. Man kann beliebig zwischen den Fenster wechseln und Daten in die Eingabefelder eingeben.

Der Toplevel-Dialog (wie auch der in Kapitel 49.2 auf Seite 557 vorgestellte Modal-Dialog) wird in der Regel innerhalb einer Prozedur erzeugt. Dabei muss man darauf achten, dass die mit den Elementen verknüpften Variablen im globalen Namensraum erzeugt werden und somit nicht identisch sind mit den lokalen Variablen innerhalb der Prozedur.

Listing 49.3: Die Variablen der Elemente sind global (Beispiel476.tcl)

```
#!/usr/bin/env wish

proc Toplevel {} {
  global Eingabe2

  set Eingabe1 "Eingabe1"
  set Eingabe2 "Eingabe2"

  toplevel .top
  ttk::entry .top.enEingabe1 -textvariable Eingabe1
  ttk::entry .top.enEingabe2 -textvariable Eingabe2
  pack .top.enEingabe1
  pack .top.enEingabe2
}

ttk::button .bt -text "Toplevel-Dialog erstellen" -command Toplevel
pack .bt
```



In den Zeilen 6 und 7 werden die beiden Variablen `Eingabe1` und `Eingabe2` mit Text vorbelegt. Beide Variablen werden in den Zeilen 10 und 11 mit Entry-Elementen verknüpft. Wenn man das Programm ausführt, sieht man, dass nur im unteren Entry-Element der Inhalt der Variable dargstellt wird. Da die mit den Elementen verknüften Variablen `Eingabe1` und `Eingabe2` zum globalen Namensraum gehören, ist die Variable `Eingabe1` des Entry-Elements nicht identisch mit der Variablen `Eingabe1` aus der Prozedur. Deshalb ist das obere Entry-Element leer. Die Variable `Eingabe2` wurde in der Zeile 4 als global festgelegt, so dass diese Variable identisch ist mit der Variable vom Entry-Element.

49.2 Modal-Dialog

Befehle:

- `grab`
- `destroy`
- `tkwait visibility`
- `tkwait window`

Oft möchte man, dass der Anwender erst in dem neuen Fenster eine Eingabe macht, bevor er in dem ersten Fenster (also dem Hauptprogramm) weiter arbeiten kann. Dies macht man mit einem Modal-Dialog.

Ein Modal-Dialog ist wie ein Toplevel-Dialog, nur dass alle anderen Fenster solange gesperrt sind, bis der Modal-Dialog beendet wurde. Wenn man Werte vom Modal-Dialog an den aufrufenden Programmteil zurückgeben will, verwendet man am besten eine globale Variable.

Tabelle 49.1: Die wichtigsten Befehle

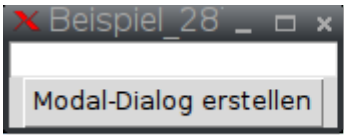
Befehl	Beschreibung
`grab set Fensternamen`	Schaltet den `modal`-Modus ein.
`grab release Fensternamen`	Schaltet den `modal`-Modus wieder aus.
`destroy Fensternamen`	Löscht das Fenster
`tkwait visibility Fensternamen`	Wartet bis das Fenster sichtbar ist

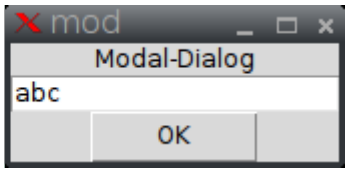
Tabelle 49.1: Die wichtigsten Befehle

Befehl	Beschreibung
`tkwait window Fenstername`	Wartet bis das Fenster geschlossen wurde
`tkwait variable Variable`	Wartet bis der Inhalt einer Variable geändert wurde

Listing 49.4: Modal-Dialog (Beispiel287.tcl)

```
1  #!/usr/bin/env wish
2
3  proc ModalDialog {} {
4    toplevel .mod
5    ttk::label .mod.lb -text "Modal-Dialog"
6    ttk::entry .mod.en
7    ttk::button .mod.bt -text "OK" -command {grab release
       .mod; destroy .mod}
8    pack .mod.lb
9    pack .mod.en
10   pack .mod.bt
11   tkwait visibility .mod
12   grab set .mod
13 }
14
15 ttk::entry .en
16 ttk::button .bt -text "Modal-Dialog erstellen" -command
     ModalDialog
17 pack .en
18 pack .bt
```



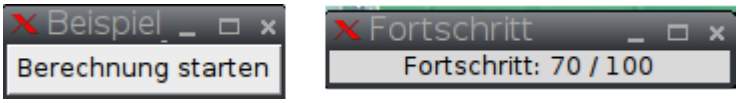


In Zeile 4 wird ein neues Toplevel-Fenster erzeugt. In Zeile 11 wird solange gewartet, bis der Modal-Dialog sichtbar ist. In Zeile 12 wird das Fenster modal gemacht, d. h. alle anderen Fenster werden deaktiviert. In Zeile 7 wird festgelegt, dass bei einem Klick auf den Button der `grab`-Befehl wieder aufgehoben und das Fenster geschlossen wird.

Listing 49.5: Fortschrittsanzeige als Modal-Dialog (Beispiel288.tcl)

```
1  #!/usr/bin/env wish
2
3  proc FortschrittAnzeigen {} {
4    set CursorAlt [. cget -cursor]
5
```

```
6    toplevel .mod
7    ttk::label .mod.lb -text ""
8    pack .mod.lb
9    tkwait visibility .mod
10   grab set .mod
11   . configure -cursor watch
12   .mod configure -cursor watch
13   wm title .mod "Fortschritt"
14   wm minsize .mod 200 1
15
16   return $CursorAlt
17 }
18
19 proc FortschrittLoeschen {CursorAlt} {
20   . configure -cursor $CursorAlt
21   .mod configure -cursor $CursorAlt
22   grab release .mod
23   destroy .mod
24 }
25
26 proc Berechnung {} {
27   set CursorAlt [FortschrittAnzeigen]
28   .mod.lb configure -text "Fortschritt:"
29   update idletasks
30
31   set Anzahl 100
32   for {set i 1} {$i <= $Anzahl } {incr i} {
33     after 100
34     if {[expr $i % 10] == 0} {
35       .mod.lb configure -text "Fortschritt: $i / $Anzahl"
36       update idletasks
37     }
38   }
39
40   FortschrittLoeschen $CursorAlt
41 }
42
43 ttk::button .bt -text "Berechnung starten" -command ↩
     Berechnung
44 pack .bt
```



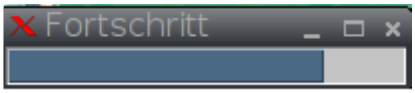
Die Prozedur in Zeile 3 erzeugt einen Modal-Dialog, der den Fortschritt anzeigt. Die Prozedur in Zeile 19 löscht den Modal-Dialog. Die Prozedur in Zeile 26 enthält exemplarisch eine (lange dauernde) Befehlsfolge.
In Zeile 4 wird das Aussehen des Mauszeigers gespeichert, bevor in den Zeilen 11 und 12 der Mauszeiger sowohl für das Hauptfenster als auch für den Modal-Dialog in eine Sanduhr geändert wird. Zeile 13 setzt den Titel des Modal-Dialogs. Zeile 14 legt die minimale Größe

des Modal-Dialogs fest. In Zeile 16 wird der alte Mauszeiger zurückgegeben, damit später beim Schließen des Modal-Dialogs der ursprüngliche Mauszeiger wieder gesetzt werden kann. In Zeile 27 wird der Modal-Dialog aufgerufen und der alte Mauszeiger gespeichert. In Zeile 29 wird die Bildschirmanzeige aktualisiert. Die Zeile 33 steht für die eigentliche Berechnung. In dem Beispiel wird lediglich eine Zehntelsekunde gewartet. In Zeile 34 wird festgelegt, dass nach jeweils zehn Schleifendurchläufen (also nach zehn Berechnungen), die Fortschrittsanzeige aktualisiert werden soll. Das %-Zeichen ist eine Rechenfunktion und gibt als Ergebnis den Rest einer Division zurück. In Zeile 40 wird der Modal-Dialog wieder gelöscht. Damit der alte Mauszeiger wieder hergestellt werden kann, wird der Wert an die Prozedur übergeben.

Listing 49.6: Fortschrittsanzeige (Progressbar) als Modal-Dialog (Beispiel289.tcl)

```
1  #!/usr/bin/env wish
2
3  proc FortschrittAnzeigen {} {
4    set CursorAlt [. cget -cursor]
5
6    toplevel .mod
7    ttk::progressbar .mod.pb -orient horizontal -maximum 100 -length 200 -value 0
8    pack .mod.pb
9    tkwait visibility .mod
10   grab set .mod
11   . configure -cursor watch
12   .mod configure -cursor watch
13   wm title .mod "Fortschritt"
14
15   return $CursorAlt
16 }
17
18 proc FortschrittLoeschen {CursorAlt} {
19   . configure -cursor $CursorAlt
20   .mod configure -cursor $CursorAlt
21   grab release .mod
22   destroy .mod
23 }
24
25 proc Berechnung {} {
26   set CursorAlt [FortschrittAnzeigen]
27   update idletasks
28
29   set Anzahl 100
30   for {set i 1} {$i <= $Anzahl } {incr i} {
31     after 100
32     if {[expr $i % 10] == 0} {
33       .mod.pb configure -value $i
34       update idletasks
35     }
36   }
37
38   FortschrittLoeschen $CursorAlt
39 }
```

```
40
41 ttk::button .bt -text "Berechnung starten" -command ↩
     Berechnung
42 pack .bt
```

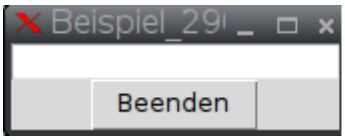


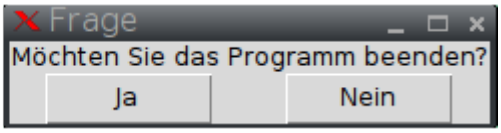
In Zeile 7 wird der Fortschrittsbalken definiert. In Zeile 33 wird der Fortschrittsbalken auf den aktuellen Wert gesetzt.

Listing 49.7: Modal-Dialog als Message-Box mit Buttons (Beispiel290.tcl)

```
1  #!/usr/bin/env wish
2
3  proc ModalDialog {Titel Text Fokus args} {
4    global GlobRueckgabewert
5
6    set GlobRueckgabewert {}
7    set AnzahlButtons [llength $args]
8
9    toplevel .mod
10   wm title .mod $Titel
11
12   ttk::label .mod.lb -text $Text
13   grid .mod.lb -row 0 -column 0 -columnspan $AnzahlButtons
14
15   set ButtonIndex 0
16   foreach Button $args {
17     ttk::button .mod.bt$ButtonIndex -text $Button -command↩
         "grab release .mod; destroy .mod; set ↩
       GlobRueckgabewert $ButtonIndex"
18     grid .mod.bt$ButtonIndex -row 1 -column $ButtonIndex
19     bind .mod.bt$ButtonIndex <Return> {%W invoke}
20     incr ButtonIndex
21   }
22
23   focus .mod.bt$Fokus
24   tkwait visibility .mod
25   grab set .mod
26 }
27
28 proc ProgrammBeenden {} {
29   global GlobRueckgabewert
30
31   ModalDialog "Frage" "Moechten Sie das Programm beenden?"↩
       1 Ja Nein
32   tkwait window .mod
33   if {$GlobRueckgabewert == 0} {
34     destroy .
35   }
36 }
```

```
37
38 set GlobRueckgabewert {}
39
40 ttk::entry .en
41 ttk::button .bt -text "Beenden" -command ProgrammBeenden
42 pack .en
43 pack .bt
```





In Zeile 3 wird die Prozedur `ModalDialog` definiert. An die Prozedur werden mehrere Parameter übergeben: zuerst der Titel, dann der Text. Der dritte Parameter `Fokus` gibt die Nummer des Buttons an, der den Fokus haben soll. Dieser Button wird ausgelöst, wenn der Anwender im Modal-Dialog nur die Return-Taste drückt. Der erste Button hat die Nummer 0, der zweite Button die Nummer 1 usw. Der vierte Parameter nimmt die Beschriftungen der einzelnen Buttons auf. In Zeile 4 wird eine globale Variable definiert, die den Index des gedrückten Buttons speichert. In den Zeilen 16 bis 21 werden so viele Buttons erzeugt, wie Beschriftungen an die Prozedur übergeben wurden (gespeichert in der Variablen `args`). Der `command`-Befehl in Zeile 17 umfasst mehrere Befehle, die durch Semikolon getrennt sind. Der Befehl `grab release .mod` hebt die Modal-Eigenschaft des Fensters auf. Der Befehl `destroy .mod` löscht den Modal-Dialog und der Befehl `set GlobRueckgabewert $ButtonIndex` speichert den Index des gedrückten Buttons, damit der aufrufende Programmteil auf die Aktion des Anwenders reagieren kann. In Zeile 18 wird jeder Button mit der Return-Taste verbunden, so dass der Anwender auch ohne Mauszeiger den Dialog bedienen kann. In Zeile 23 wird der Fokus den Button gesetzt. In Zeile 24 wartet das Programm, bis der Modal-Dialog sichtbar wird. In Zeile 25 wird der Dialog modal gemacht, das heißt, der Anwender kann nur noch in diesem Dialog eine Aktion ausführen. Die Prozedur in Zeile 28 wird aufgerufen, wenn der Anwender im Hauptfenster auf den Beenden-Button drückt. In Zeile 31 wird der Modal-Dialog aufgerufen. In Zeile 32 wartet das Programm, bis der Modal-Dialog beendet wurde. In Zeile 33 wird der Rückgabewert aus dem Modal-Dialog ausgewertet. In diesem Fall wird das Programm beendet (Zeile 34), wenn der Button „ja“ (dieser hat den Index 0) gedrückt wurde.

50 TkTable

Das TkTable-Element wird für die Darstellung von Tabellen verwendet. Eventuell muss man das Paket `tk-table` nachinstallieren. Es sollte im Repository der jeweiligen Linux-Distribution vorhanden sein. Man kann das TkTable-Element mit einer Matrix verknüpfen. Ein Beispiel hierzu findet man in Kapitel 19 auf Seite 223 zur Matrix.

Tabelle 50.1: Die wichtigsten Optionen

Option	Beschreibung
`-state normal` `-state disabled`	Legt fest, ob die Tabelle editierbar ist.
`-rows Anzahl`	Anzahl Zeilen. Bestimmt auch die Länge der Scroll-Leiste. Wenn nichts angegeben wird, ist der Wert 10.
`-cols Anzahl`	Anzahl Spalten. Bestimmt auch die Länge der Scroll-Leiste. Wenn nichts angegeben wird, ist der Wert 10.
`-titlerows Anzahl`	Anzahl der Zeilen, die für die Spaltenbeschriftung verwendet werden. Wenn nichts angegeben wird, ist der Wert 0.
`-titlecols Anzahl`	Anzahl der Spalte, die für die Zeilenbeschriftung verwendet werden. Wenn nichts angegeben wird, ist der Wert 0.
`-height Anzahl`	Höhe der Tabelle in Zeilenanzahl (= sichtbare Zeilen). Wenn nichts angegeben wird, wird die Tabelle so groß dargestellt, dass alle Zeilen hineinpassen.
`-width Anzahl`	Breite der Tabelle in Spaltenanzahl (= sichtbare Spalten). Wenn nichts angegeben wird, wird die Tabelle so groß dargestellt, dass alle Spalten hineinpassen.

Tabelle 50.1: Die wichtigsten Optionen

Option	Beschreibung
`-rowheight Anzahl`	Höhe der Zeile in Zeilenanzahl. Dies ist der Default-Wert für alle Zeilen. Wenn nichts angegegen wird, ist der Wert 1.
`-colwidth Anzahl`	Breite der Spalten in Zeichenanzahl. Dies ist der Default-Wert für alle Spalten. Wenn nichts angegeben wird, ist der Wert 10.
`-maxheight Pixel`	Maximale Tabellenhöhe in Pixel. Wenn nichts angegeben wird, ist der Wert 600.
`-maxwidth Pixel`	Maximale Tabellenbreite in Pixel. Wenn nichts angegeben wird, ist der Wert 800.
`-padx Pixel`	Waagrechter Abstand des Zelleninhalts vom Rand in Pixel.
`-pady Pixel`	Senkrechter Abstand des Zelleninhalts vom Rand in Pixel.

`-anchor w`
`-anchor e`
`-anchor c`
	Formatiert die Ausrichtung der Spalten.
	`w`=linksbündig (west)
	`e`=rechtsbündig (east)
	`c`=zentriert (center)

`-justify center`
`-justify left`
`-justify right`
	Legt die Ausrichtung bei mehrzeiligem Text in einer Zelle fest. Wenn nichts angegeben wird, ist der Wert `left`.

`-multiline 0` `-multiline 1`	Legt fest, ob eine Zelle mehrzeilig sein darf. `0` = nein, `1` = ja. Wenn nichts angegeben wird, ist der Wert 1.
`-selectmode browse` `-selectmode extended`	Legt fest, ob ein oder mehrere Elemente ausgewählt werden können. `browse` = es kann genau ein Element ausgewählt werden `extended` = es können mehrere Elemente (mittels der `Strg`-Taste) ausgewählt werden

Tabelle 50.1: Die wichtigsten Optionen

Option	Beschreibung
`-variable Variable`	Benennt die Variable, die die Zelleneinträge enthält. Die Variable ist ein Array in der Form Variable(Zeile,Spalte). Die erste Zelle links oben in der Tabelle hat den Index (0,0).
`-font Schrift`	Schriftart und -größe

In der Tabelle steuert man mit folgenden Tasten:

Tabelle 50.2: Tastensteuerung

Taste	Beschreibung
`Cursor`	links, rechts, hoch, runter
`Strg+Pos1`	springt zur Zelle links oben
`Strg+Ende`	springt zur Zelle rechts unten
`Strg+Cursor links`	bewegt den Cursor innerhalb der Zelle nach links
`Strg+Cursor rechts`	bewegt den Cursor innerhalb der Zelle nach rechts
`Ziehen der Spalten- oder Zeilenbegrenzung mit gedrückter linker Maustaste`	Verändert die Spaltenbreite bzw. Zeilenhöhe
`Umschaltung+Cursor`	Selektiert einen zusammenhängenden Zellbereich
`Linke Maustaste drücken und Maus über Zellen ziehen`	Selektiert Zellen
`Strg+Mausklick auf Zelle`	Fügt eine Zelle der Selektion hinzu
`Mausklick auf Spalten- oder Zeilenbeschriftung`	Selektiert die gesamte Spalte (bzw. Zeile). Voraussetzung: `-selectmode extended` wurde eingestellt.

Die wichtigsten Tabellen-Befehle sind:

Tabelle 50.3: Die wichtigsten Tabellenbefehle

Befehl	Beschreibung
`[.table index topleft]`	Gibt die Zelle an, die links oben im sichtbaren Bereich liegt
`[.table index bottomright]`	Gibt die Zelle an, die rechts unten im sichtbaren Bereich liegt
`[.table index end]`	Gibt die Zelle rechts unten am Tabellenende an
`[.table index origin]`	Gibt die Zelle an, die am weitesten links oben liegt und editierbar ist
`[.table xview Spalte]`	Setzt die Zelle links oben auf die angegebene Spalte
`[.table yview Zeile]`	Setzt die Zelle links oben auf die angegebene Zeile
`[.table curselection]`	Liefert alle Indizes der Selektion
`[.table get rows]`	Liefert die Anzahl der Zeilen
`[.table get cols]`	Liefert die Anzahl der Spalten
`[.table get Index]`	Liefert den Tabellenwert, passend zum Index. Der Index wird in der Form `Zeile,Spalte` angegeben.
`[.table get IndexVon IndexBis]`	Liefert die Tabellenwerte eines Bereichs (z. B. einer Zeile oder Spalte)
`.table set row Index Liste`	Ändert den Zellinhalt in einer Zeile (aber der Zelle mit dem Index). Die Liste enthält den neuen Zellinhalt.
`.table set col Index Liste`	Ändert den Zellinhalt in einer Spalte (aber der Spalte mit dem Index). Die Liste enthält den neuen Zellinhalt.
`.table insert rows Zeile Anzahl`	Fügt Zeilen ein. Wenn die Anzahl negativ ist, werden die Zeilen oberhalb eingefügt.
`.table insert cols Spalte Anzahl`	Fügt Spalten ein. Wenn die Anzahl negativ ist, werden die Spalten links eingefügt.

Tabelle 50.3: Die wichtigsten Tabellenbefehle

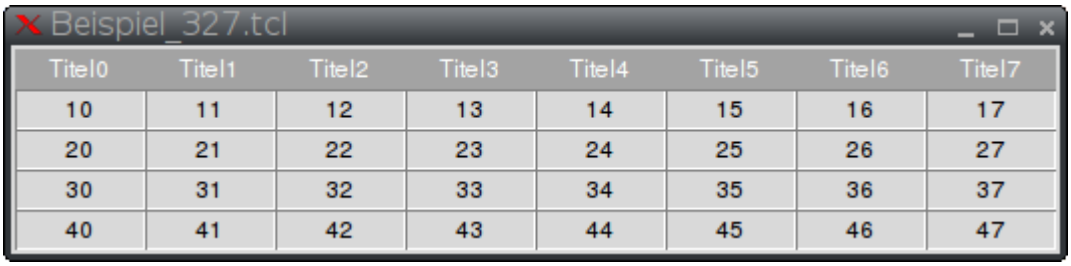
Befehl	Beschreibung
`.table delete rows Zeile Anzahl`	Löscht Zeilen. Wenn die Anzahl negativ ist, werden die Zeilen oberhalb gelöscht.
`.table delete cols Spalte Anzahl`	Löscht Spalten. Wenn die Anzahl negativ ist, werden die Spalten links gelöscht.
`.table width Spalte Breite`	Legt die Spaltenbreite fest
`.table configure Eigenschaft Wert`	Ändert die Eigenschaft (z. B. Zeilen- oder Spaltenanzahl)
`.table tag col tagName Spalte`	Gibt der Spalte einen (internen) Namen (ein sogenannter `tag`), so dass man die Spalte individuell formatieren kann. Der Name muss mit einem Kleinbuchstaben beginnen.
`.table tag row tagName Zeile`	Gibt der Zeile einen (internen) Namen (ein sogenannter `tag`), so dass man die Zeile individuell formatieren kann. Der Name muss mit einem Kleinbuchstaben beginnen.
`.table tag cell tagName Zeile,Spalte`	Gibt der Zelle einen (internen) Namen (ein sogenannter `tag`), so dass man die Zelle individuell formatieren kann. Der Name muss mit einem Kleinbuchstaben beginnen.
`.table tag configure tagName -anchor Ausrichtung`	Formatiert die Ausrichtung des mit `tagName` definierten Bereichs (Zelle, Spalte, Zeile). `w`=linksbündig (west) `e`=rechtsbündig (east) `c`=zentriert (center)
`.table tag configure tagName -font "Schriftart Größe Stil"`	Formatiert die Schrift für den mit `tagName` definierten Bereich (Zelle, Spalte, Zeile)
`.table tag configure tagName -background Farbe`	Formatiert die Hintergrundfarbe für den mit `tagName` definierten Bereich (Zelle, Spalte, Zeile)

Tabelle 50.3: Die wichtigsten Tabellenbefehle

Befehl	Beschreibung
`.table tag configure tagName -foreground Farbe`	Formatiert die Vordergrundfarbe für den mit `tagName` definierten Bereich (Zelle, Spalte, Zeile)

Listing 50.1: Tabelle ohne Scroll-Leisten (Beispiel327.tcl)

```
1  #!/usr/bin/env wish
2
3  package require Tktable
4
5  table .table \
6    -rows 5 \
7    -cols 8 \
8    -titlerows 1 \
9    -titlecols 0 \
10   -height 5 \
11   -width 25 \
12   -rowheight 1 \
13   -colwidth 9 \
14   -selectmode extended \
15   -variable Wert \
16
17 pack .table -side right -fill both -expand 1
18
19 for {set Zeile 0} {$Zeile <= 4} {incr Zeile} {
20   for {set Spalte 0} {$Spalte <= 7} {incr Spalte} {
21     if {$Zeile == 0} {
22       set Wert($Zeile,$Spalte) Titel$Spalte
23     } else {
24       set Wert($Zeile,$Spalte) $Zeile$Spalte
25     }
26   }
27 }
```

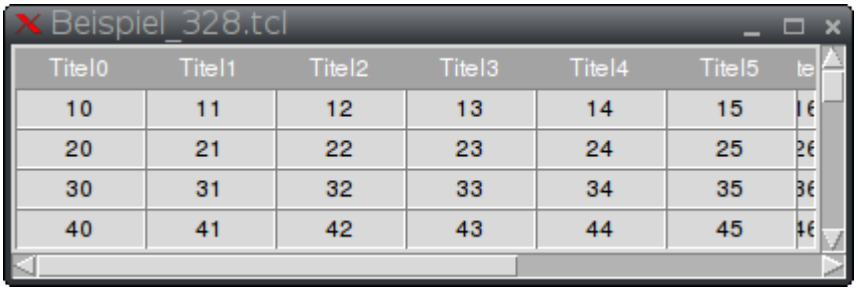


In Zeile 3 wird das Paket `Tktable` eingebunden. In den Zeilen 5 bis 15 wird die Tabelle definiert. Der Schrägstrich `text` am Zeilenende bedeutet, dass der Befehl in der nächsten Zeile weiter geht. Dadurch wird die Darstellung der Optionen übersichtlicher. Zeile 17

zeigt die Tabelle an. Die Zeilen 19 bis 27 füllen die Tabelle mit Werten. Man sieht, dass die mit der Tabelle verknüpfte Variable `Wert` ein Array in der Form Variable(Zeile,Spalte) ist und die Zelle links oben den Index (0,0) hat.

Listing 50.2: Tabelle mit Scroll-Leisten (Beispiel328.tcl)

```
1  #!/usr/bin/env wish
2
3  package require Tktable
4
5  ttk::frame .fr
6
7  table .fr.table \
8    -rows 20 \
9    -cols 10 \
10   -titlerows 1 \
11   -titlecols 0 \
12   -height 5 \
13   -width 25 \
14   -rowheight 1 \
15   -colwidth 9 \
16   -maxheight 100 \
17   -maxwidth 400 \
18   -selectmode extended \
19   -variable Wert \
20   -xscrollcommand {.fr.xscroll set} \
21   -yscrollcommand {.fr.yscroll set}
22
23 ttk::scrollbar .fr.xscroll -command {.fr.table xview} ↩
     -orient horizontal
24 ttk::scrollbar .fr.yscroll -command {.fr.table yview}
25
26 pack .fr -fill both -expand 1
27 pack .fr.xscroll -side bottom -fill x
28 pack .fr.yscroll -side right -fill y
29 pack .fr.table -side right -fill both -expand 1
30
31 for {set Zeile 0} {$Zeile <= 19} {incr Zeile} {
32   for {set Spalte 0} {$Spalte <= 9} {incr Spalte} {
33     if {$Zeile == 0} {
34       set Wert($Zeile,$Spalte) Titel$Spalte
35     } else {
36       set Wert($Zeile,$Spalte) $Zeile$Spalte
37     }
38   }
39 }
```

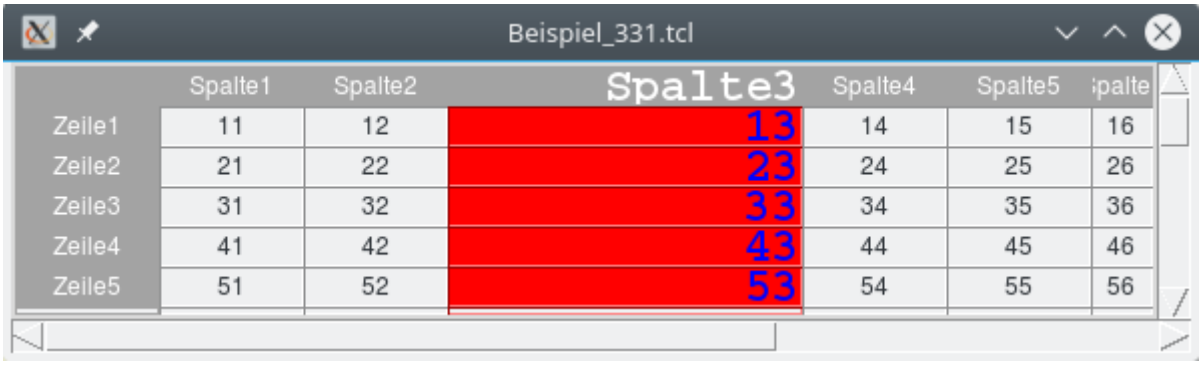


In Zeile 5 wird ein Rahmen (Frame) definiert, der die Tabelle mit den beiden Scroll-Leisten aufnimmt. In den Zeilen 7 bis 21 wird die Tabelle beschrieben. Dabei wird in den Zeilen 20 und 21 die Verknüpfung mit den Scroll-Leisten festgelegt. Die Zeilen 23 und 24 erstellen die Scroll-Leisten. In den Zeilen 26 bis 29 wird die Tabelle mit den Scroll-Leisten dargestellt. In den Zeilen 31 bis 39 wird die Tabelle mit Werten gefüllt.

Listing 50.3: Tabelle mit Spalten- und Zeilenüberschrift sowie individueller Formatierung (Beispiel331.tcl)

```
1  #!/usr/bin/env wish
2
3  package require Tktable
4
5  ttk::frame .fr
6
7  table .fr.table \
8    -rows 20 \
9    -cols 10 \
10   -titlerows 1 \
11   -titlecols 1 \
12   -height 5 \
13   -width 25 \
14   -rowheight 1 \
15   -colwidth 10 \
16   -maxheight 100 \
17   -maxwidth 400 \
18   -anchor c \
19   -selectmode extended \
20   -variable Wert \
21   -xscrollcommand {.fr.xscroll set} \
22   -yscrollcommand {.fr.yscroll set}
23
24 for {set Zeile 0} {$Zeile <= 19} {incr Zeile} {
25   for {set Spalte 0} {$Spalte <= 9} {incr Spalte} {
26     if {$Spalte == 0} {
27       if {$Zeile > 0} {
28         set Wert($Zeile,$Spalte) Zeile$Zeile
29       }
30     } else {
31       if {$Zeile == 0} {
32         set Wert($Zeile,$Spalte) Spalte$Spalte
33       } else {
34         set Wert($Zeile,$Spalte) $Zeile$Spalte
```

```
35          }
36       }
37    }
38 }
39
40 .fr.table width 3 25
41
42 .fr.table tag col spalteA 3
43 .fr.table tag configure spalteA -anchor e
44 .fr.table tag configure spalteA -font {courier 18 bold}
45 .fr.table tag configure spalteA -background red
46 .fr.table tag configure spalteA -foreground blue
47
48 ttk::scrollbar .fr.xscroll -command {.fr.table xview} ↩
      -orient horizontal
49 ttk::scrollbar .fr.yscroll -command {.fr.table yview}
50
51 pack .fr -fill both -expand 1
52 pack .fr.xscroll -side bottom -fill x
53 pack .fr.yscroll -side right -fill y
54 pack .fr.table -side right -fill both -expand 1
```

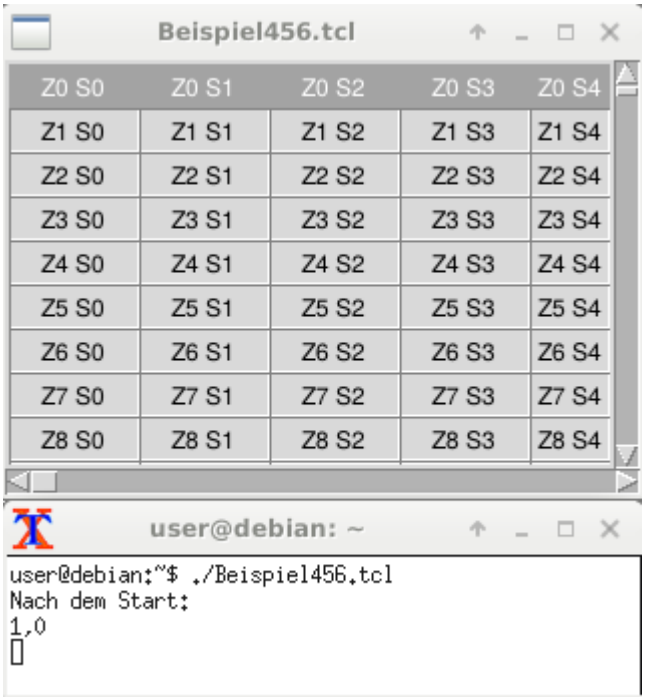


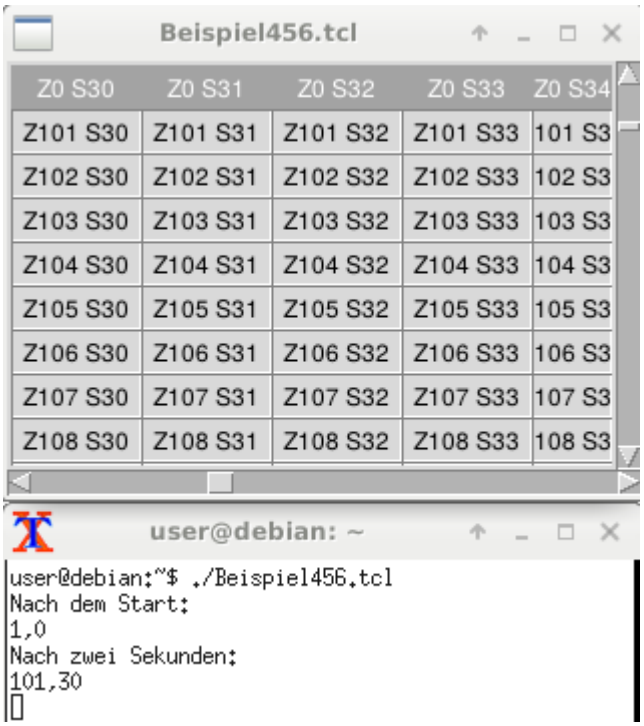
In den Zeilen 10 und 11 wird festgelegt, dass die erste Zeile Spaltenüberschriften und die erste Spalte Zeilenüberschriften hat. In Zeile 18 werden die Zellinhalte zentriert dargestellt. In Zeile 40 wird die Spaltenbreite für die vierte Spalte (Index = `3`) auf 25 Zeichen festgelegt. In Zeile 42 erhält die vierte Spalte (Index = `3`) eine Bezeichnung (ein sogenannter `tag`), über die die Spalte in den folgenden Zeilen 43 bis 46 angesprochen und formatiert werden kann. In Zeile 43 wird die Spalte rechtsbündig (`e` = east = Osten) ausgerichtet. In Zeile 44 wird die Schriftart für die Spalte geändert. In Zeile 45 wird der Hintergrund rot eingestellt. In Zeile 46 wird der Vordergrund blau eingestellt.

Listing 50.4: Sichtbare Zelle links oben ändern (Beispiel456.tcl)

```
1 #!/usr/bin/env wish
2
3 package require Tktable
4
5 ttk::frame .fr
```

```

table .fr.table \
  -rows 1000 \
  -cols 100 \
  -titlerows 1 \
  -titlecols 0 \
  -height 20 \
  -width 25 \
  -rowheight 1 \
  -colwidth 9 \
  -maxheight 200 \
  -maxwidth 300 \
  -selectmode extended \
  -variable Wert \
  -xscrollcommand {.fr.xscroll set} \
  -yscrollcommand {.fr.yscroll set} \
  -state disabled

ttk::scrollbar .fr.xscroll -command {.fr.table xview} \
   -orient horizontal
ttk::scrollbar .fr.yscroll -command {.fr.table yview}

pack .fr -fill both -expand 1
pack .fr.xscroll -side bottom -fill x
pack .fr.yscroll -side right -fill y
pack .fr.table -side right -fill both -expand 1

for {set Zeile 0} {$Zeile <= 1000} {incr Zeile} {
  for {set Spalte 0} {$Spalte <= 100} {incr Spalte} {
    set Wert($Zeile,$Spalte) "Z$Zeile S$Spalte"
  }
}

.fr.table xview 0
.fr.table yview 0
puts "Nach dem Start:"
puts [.fr.table index topleft]
update idletasks

after 2000

.fr.table xview 30
.fr.table yview 100
update idletasks
puts "Nach zwei Sekunden:"
puts [.fr.table index topleft]
```





In den Zeilen 38 und 39 wird festgelegt, welche Zelle links oben in der Tabelle sichtbar sein soll. In diesem Fall die Zelle mit dem Zeilenindex 0 und Spaltenindex 0. Dabei werden die Titelzeilen und Titelspalten nicht mitgezählt. In dem Beispiel gibt es eine Titelzeile. Somit repräsentiert der Zeilenindex `0` im Befehl `.fr.table yview 0` die Zeile mit dem Tabellenindex `1`. Nach zwei Sekunden wird die sichtbare Zelle links oben verändert. Dies

erfolgt in den Zeilen 46 und 47.

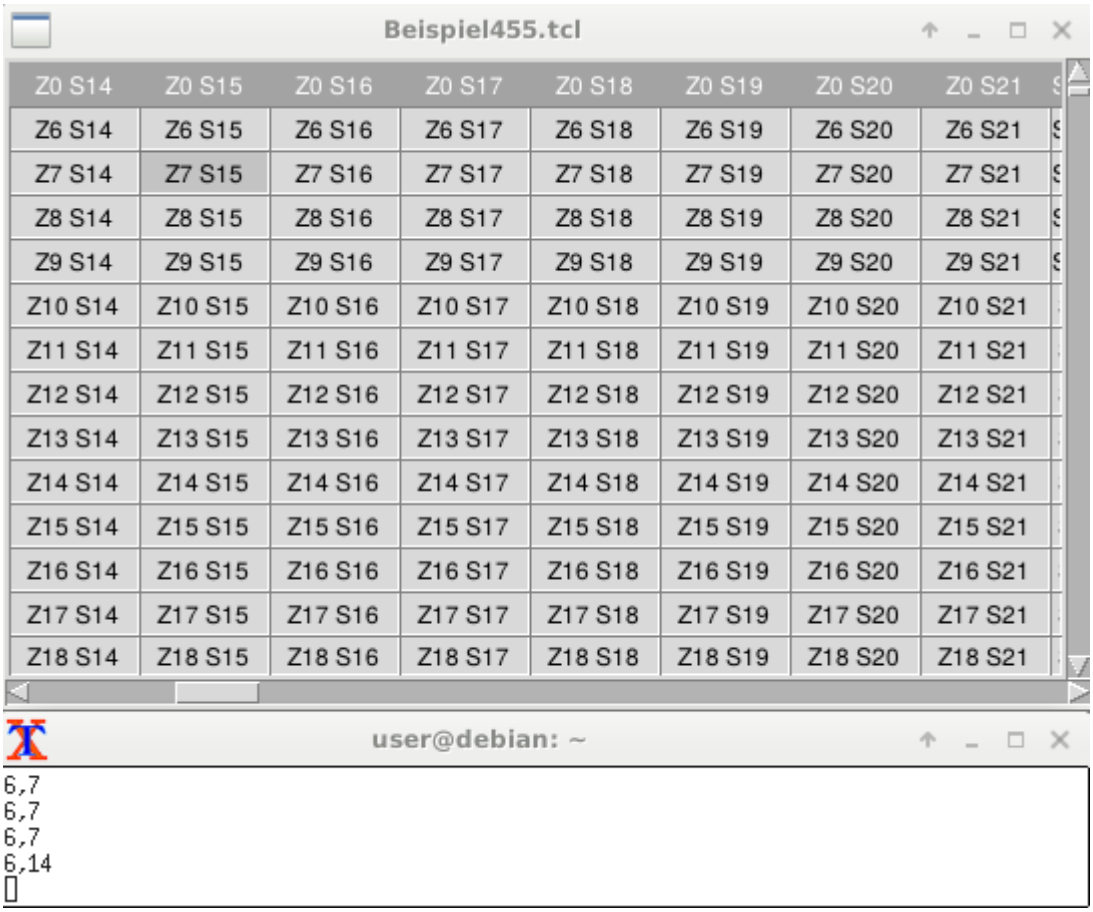
50.1 Maus- und Tastaturereignisse

Listing 50.5: Maus- und Tastaturereignisse (Beispiel455.tcl)

```
1  #!/usr/bin/env wish
2
3  package require Tktable
4
5  proc NachObenScrollen {} {
6          set Koordinaten [split [.fr.table index topleft] "
           ,"]
7          set Zeile [lindex $Koordinaten 0]
8
9          if {$Zeile >= 10} {
10                 incr Zeile -10
11         } else {
12                 set Zeile 0
13         }
14
15         .fr.table yview $Zeile
16         update idletasks
17         puts [.fr.table index topleft]
18 }
19
20 proc NachUntenScrollen {} {
21         set Koordinaten [split [.fr.table index topleft] "
           ,"]
22         set Zeile [lindex $Koordinaten 0]
23
24         incr Zeile 10
25
26         .fr.table yview $Zeile
27         update idletasks
28         puts [.fr.table index topleft]
29 }
30
31 ttk::frame .fr
32
33 table .fr.table \
34   -rows 1000 \
35   -cols 100 \
36   -titlerows 1 \
37   -titlecols 0 \
38   -height 20 \
39   -width 25 \
40   -rowheight 1 \
41   -colwidth 9 \
42   -maxheight 300 \
43   -maxwidth 400 \
44   -selectmode extended \
```

```
45   -variable Wert \
46   -xscrollcommand {.fr.xscroll set} \
47   -yscrollcommand {.fr.yscroll set} \
48   -state disabled
49
50 ttk::scrollbar .fr.xscroll -command {.fr.table xview} ↩
     -orient horizontal
51 ttk::scrollbar .fr.yscroll -command {.fr.table yview}
52
53 pack .fr -fill both -expand 1
54 pack .fr.xscroll -side bottom -fill x
55 pack .fr.yscroll -side right -fill y
56 pack .fr.table -side right -fill both -expand 1
57
58 for {set Zeile 0} {$Zeile <= 1000} {incr Zeile} {
59   for {set Spalte 0} {$Spalte <= 100} {incr Spalte} {
60     set Wert($Zeile,$Spalte) "Z$Zeile S$Spalte"
61   }
62 }
63
64 .fr.table xview 0
65 .fr.table yview 0
66 update idletasks
67 puts [.fr.table index topleft]
68
69 # Bindings
70 bind .fr.yscroll <ButtonRelease-1> {puts [.fr.table index ↩
     topleft]}
71 bind .fr.xscroll <ButtonRelease-1> {puts [.fr.table index ↩
     topleft]}
72 bind .fr.table <KeyRelease-Prior> {puts [.fr.table index ↩
     topleft]}
73 bind .fr.table <KeyRelease-Next> {puts [.fr.table index ↩
     topleft]}
74
75 bind .fr.table <KeyRelease-Up> {puts [.fr.table index ↩
     topleft]}
76 bind .fr.table <KeyRelease-Down> {puts [.fr.table index ↩
     topleft]}
77 bind .fr.table <KeyRelease-Left> {puts [.fr.table index ↩
     topleft]}
78 bind .fr.table <KeyRelease-Right> {puts [.fr.table index ↩
     topleft]}
79
80 bind .fr.table <KeyRelease-Home> {puts [.fr.table index ↩
     topleft]}
81 bind .fr.table <KeyRelease-End> {puts [.fr.table index ↩
     topleft]}
82
83 # Scrollrad der Maus
84 bind .fr.table <Button-4> {NachObenScrollen}
85 bind .fr.table <Button-5> {NachUntenScrollen}
86
```

```
87 # Fokus auf die Tabelle setzen, damit die Tabelle direkt ↩
     auf die Tasten reagiert
88 focus .fr.table
```

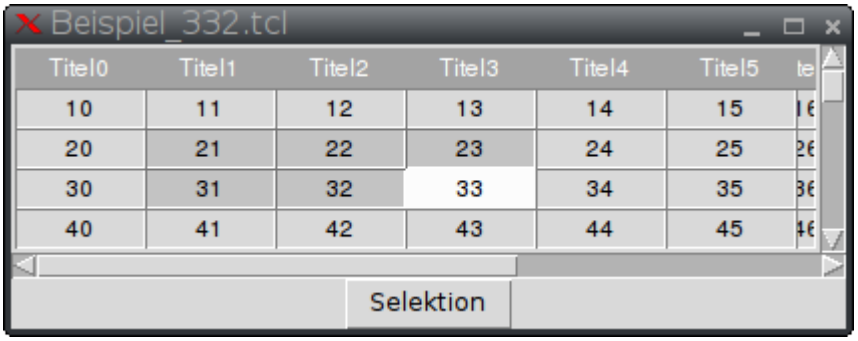


Das TkTable-Element verfügt über verschiedene Tastaturereignisse, die den sichtbaren Tabellenteil beeinflussen. Es kann erforderlich sein, dass das Programm erfährt, welche Zelle links oben durch ein solches Ereignis sichtbar geworden ist. Dazu definiert man die verschiedenen Tastaturereignisse und verknüpft sie mit der Tabelle (Zeilen 70 bis 81). Immer wenn eine Taste gedrückt und wieder losgelassen wird, ändert sich einerseits der sichtbare Tabellenausschnitt, andererseits liefert das Tastatur-Binding die Zellkoordinate links oben.
Die Zeilen 84 und 85 legen fest, wie auf das Drehen des Mausrads reagiert werden soll. Man muss beachten, dass das Drehen des Mausrads kein vordefiniertes Tabellenereignis ist, so dass sich der sichtbare Tabellenausschnitt nicht verändert. Deshalb wurden die beiden Prozeduren `NachObenScrollen` und `NachUntenScrollen` definiert, die den Tabellenausschnitt um 10 Zeilen nach oben oder unten verschieben.

50.2 Selektion auswerten und Tabelle ändern

Listing 50.6: Selektierte Werte auswerten (Beispiel332.tcl)

```
#!/usr/bin/env wish

package require Tktable

proc Selektion {Tabelle} {
  foreach Index [$Tabelle curselection] {
    set Wert [$Tabelle get $Index]
    puts "Zelle $Index hat den Wert $Wert"
  }
}

ttk::frame .fr

table .fr.table \
  -rows 20 \
  -cols 10 \
  -titlerows 1 \
  -titlecols 0 \
  -height 5 \
  -width 25 \
  -rowheight 1 \
  -colwidth 9 \
  -maxheight 100 \
  -maxwidth 400 \
  -selectmode extended \
  -variable Wert \
  -xscrollcommand {.fr.xscroll set} \
  -yscrollcommand {.fr.yscroll set}

ttk::scrollbar .fr.xscroll -command {.fr.table xview} -orient horizontal
ttk::scrollbar .fr.yscroll -command {.fr.table yview}

ttk::button .bt -text "Selektion" -command {Selektion .fr.table}

pack .fr -fill both -expand 1
pack .fr.xscroll -side bottom -fill x
pack .fr.yscroll -side right -fill y
pack .fr.table -side right -fill both -expand 1
pack .bt

for {set Zeile 0} {$Zeile <= 19} {incr Zeile} {
  for {set Spalte 0} {$Spalte <= 9} {incr Spalte} {
    if {$Zeile == 0} {
      set Wert($Zeile,$Spalte) Titel$Spalte
    } else {
      set Wert($Zeile,$Spalte) $Zeile$Spalte
    }
  }
}
```



In den Zeilen 5 bis 10 wird die Prozedur `Selektion` definiert. Sie zeigt den Index und den zugehörigen Zellenwert an. In Zeile 33 wird ein Button erzeugt, der die Prozedur aufruft.

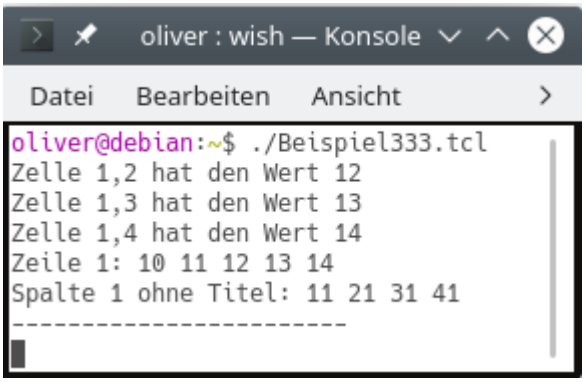
Listing 50.7: Zellen auswerten, ändern und einfügen (Beispiel333.tcl)

```
1  #!/usr/bin/env wish
2
3  package require Tktable
4
5  proc Anzeigen {Tabelle} {
6    global Tabelleninhalt
7
8    puts "Zelle 1,2 hat den Wert $Tabelleninhalt(1,2)"
9
10   set Index "1,3"
11   set Wert [$Tabelle get $Index]
12   puts "Zelle $Index hat den Wert $Wert"
13
14   set Wert [$Tabelle get 1,4]
15   puts "Zelle 1,4 hat den Wert $Wert"
16
17   set LetzteSpalte [expr [$Tabelle cget -cols] -1]
18   set Werte [$Tabelle get 1,0 1,$LetzteSpalte]
19   puts "Zeile 1: $Werte"
20
21   set LetzteZeile [expr [$Tabelle cget -rows] -1]
22   set Werte [$Tabelle get 1,1 $LetzteZeile,1]
23   puts "Spalte 1 ohne Titel: $Werte"
24   puts "------------------------"
25 }
26
```

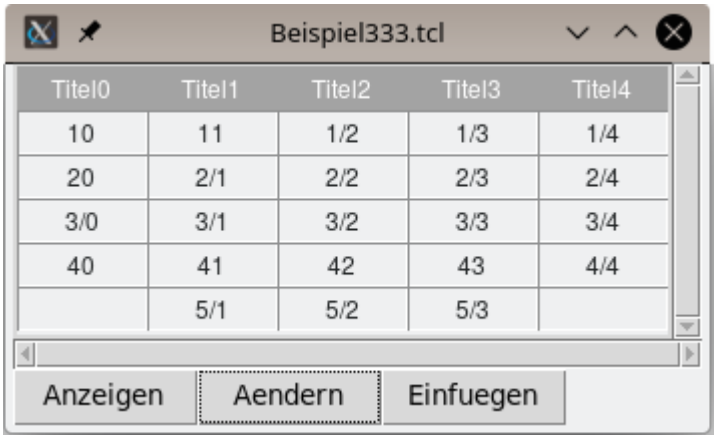
```
27 proc Aendern {Tabelle} {
28   global Tabelleninhalt
29
30   set Tabelleninhalt(1,2) "1/2"
31
32   $Tabelle set 1,3 "1/3"
33
34   $Tabelle set row 2,1 {"2/1" "2/2" "2/3"}
35
36   set Liste {"3/0" "3/1" "3/2" "3/3" "3/4"}
37   $Tabelle set row 3,0 $Liste
38
39   $Tabelle set col 1,5 {"1/4" "2/4" "3/4" "4/4"}
40
41   $Tabelle configure -rows 6
42   $Tabelle set row 5,1 {"5/1" "5/2" "5/3"}
43 }
44
45 proc Einfuegen {Tabelle} {
46   global Tabelleninhalt
47
48   $Tabelle configure -rows 6
49   $Tabelle set row 5,1 {"5/1" "5/2" "5/3"}
50
51   $Tabelle configure -rows 7
52   set Tabelleninhalt(6,0) "6/0"
53   set Tabelleninhalt(6,1) "6/1"
54
55   $Tabelle configure -cols 6
56   $Tabelle set col 1,6 {"1/5" "2/5" "3/5" "4/5" "5/5" "6/5
       "}
57 }
58
59 ttk::frame .fr
60
61 table .fr.table \
62   -rows 5 \
63   -cols 5 \
64   -titlerows 1 \
65   -titlecols 0 \
66   -height 5 \
67   -width 25 \
68   -rowheight 1 \
69   -colwidth 9 \
70   -maxheight 100 \
71   -maxwidth 400 \
72   -selectmode extended \
73   -variable Tabelleninhalt \
74   -xscrollcommand {.fr.xscroll set} \
75   -yscrollcommand {.fr.yscroll set}
76
77 ttk::scrollbar .fr.xscroll -command {.fr.table xview}
     -orient horizontal
```

```
78 ttk::scrollbar .fr.yscroll -command {.fr.table yview}
79
80 ttk::button .btAnzeigen -text "Anzeigen" -command {
     Anzeigen .fr.table}
81 ttk::button .btAendern -text "Aendern" -command {Aendern
     .fr.table}
82 ttk::button .btEinfuegen -text "Einfuegen" -command {
     Einfuegen .fr.table}
83
84 pack .fr -fill both -expand 1
85 pack .fr.xscroll -side bottom -fill x
86 pack .fr.yscroll -side right -fill y
87 pack .fr.table -side right -fill both -expand 1
88 pack .btAnzeigen -side left
89 pack .btAendern -side left
90 pack .btEinfuegen -side left
91
92 for {set Zeile 0} {$Zeile <= 4} {incr Zeile} {
93   for {set Spalte 0} {$Spalte <= 4} {incr Spalte} {
94     if {$Zeile == 0} {
95       set Tabelleninhalt($Zeile,$Spalte) Titel$Spalte
96     } else {
97       set Tabelleninhalt($Zeile,$Spalte) $Zeile$Spalte
98     }
99   }
100 }
```

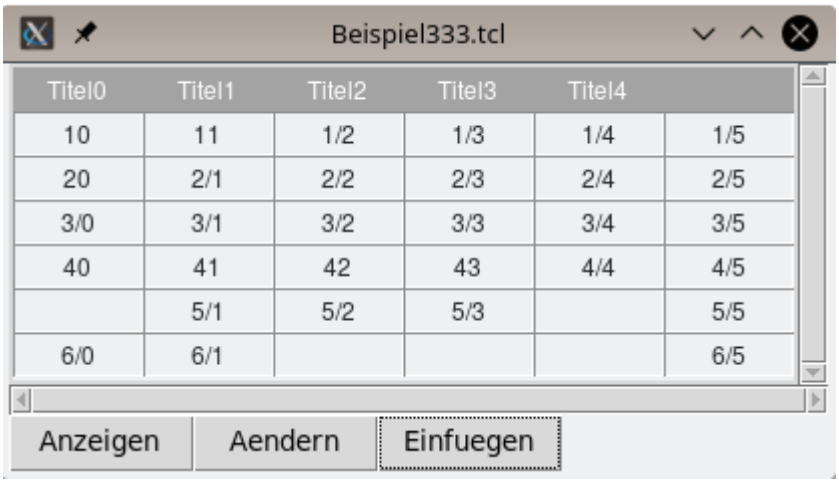
Nach einem Klick auf den Button `Anzeigen`:



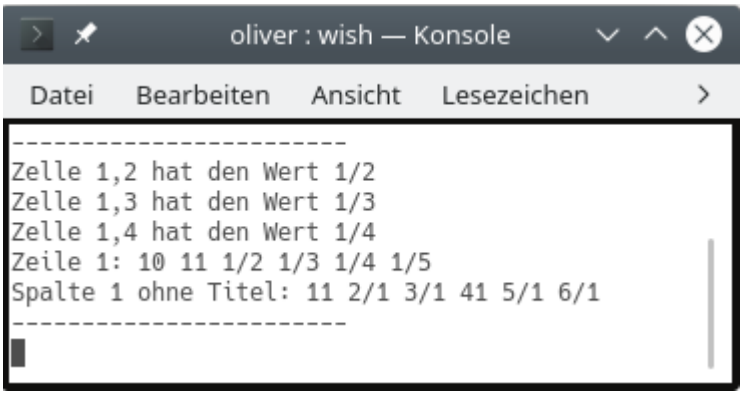
Nach einem Klick auf den Button `Ändern`:



Nach einem Klick auf den Button `Einfügen`:



Nach einem Klick auf den Button `Anzeigen`:



Das Beispiel zeigt verschiedene Möglichkeiten Zellen, Zeilen und Spalten auszulesen, zu ändern oder einzufügen. Das `TkTable`-Element bekommt in Zeile 58 den Namen `.fr.table` und wird in Zeile 70 mit der (globalen) Variablen `Tabelleninhalt` verknüpft. Diese Variable ist eine Array-Variable in der Form `Tabelleninhalt(Zeile,Spalte)`. Wie üblich haben die erste Zeile und die erste Spalte den Index 0. Die Zellen in der Tabelle können auf zwei Arten angesprochen werden: entweder über die Variable `Tabelleninhalt` oder über den Element-Namen `.fr.table`.

In den Zeilen 5 bis 25 finden Sie die Prozedur zum Anzeigen von Zellinhalten. In Zeile 8 wird der Inhalt einer Zelle über die Variable ermittelt, in Zeile 11 über den Element-Namen. Den Zellenindex kann man sowohl über eine Variable festlegen (Varaiable `Index` in den Zeilen 10 und 11) oder direkt angeben (Zeile 14). In Zeile 17 wird der Index der letzten Spalte bestimmt. In Zeile 18 werden die Zellinhalte der gesamten Zeile ermittelt. Das Ergebnis ist eine Liste, die in der Variablen `Werte` gespeichert wird. In den Zeilen 21 und 22 wird analog eine gesamte Spalte ausgelesen.

In den Zeilen 27 bis 43 sehen Sie die Prozedur, die einzelne Zellinhalte ändert. Auch hierbei gibt es verschiedene Möglichkeiten die Zellen anzusprechen. In Zeile 30 wird die Variable `Tabelleninhalt` geändert, in Zeile 32 erfolgt die Änderung über den Element-Namen. In Zeile 34 werden mehrere, fortlaufende Zellen in einer Zeile geändert.. Die Änderung beginnt in der Zelle `2,1` und überschreibt drei Zellen. Die neuen Zellinhalte werden als Liste übergeben. In den Zeilen 36 und 37 werden ebenfalls Zellen in einer Zeile geändert. Dabei enthält eine Listen-Variable den neuen Zellinhalt. In Zeile 39 erfolgt das gleiche für eine Spalte.

In den Zeilen 42 bis 54 werden neue Zeilen bzw. Spalten am Tabellenende eingefügt. Zuerst wird in Zeile 45 die Anzahl der Zeilen um eins erhöht. In Zeile 46 werden einige Zellen in der neuen Zeile gefüllt. Dabei wird die Tabelle über ihren Element-Namen angesprochen. In den Zeilen 48 bis 50 werden ebenfalls neue Zellen eingefügt. Jetzt wird aber die Variable `Tabelleninhalt` verwendet. In den Zeilen 52 bis 53 wird eine Spalte eingefügt.

Listing 50.8: Zeilen und Spalten einfügen und löschen (Beispiel335.tcl)

```
1 #!/usr/bin/env wish
2
3 package require Tktable
4
5 proc ZeileEinfuegen {Tabelle Zeile Anzahl} {
6   $Tabelle insert rows $Zeile $Anzahl
```

```
7  }
8
9  proc SpalteEinfuegen {Tabelle Spalte Anzahl} {
10   $Tabelle insert cols $Spalte $Anzahl
11 }
12
13 proc ZeileLoeschen {Tabelle Zeile Anzahl} {
14   $Tabelle delete rows $Zeile $Anzahl
15 }
16
17 proc SpalteLoeschen {Tabelle Spalte Anzahl} {
18   $Tabelle delete cols $Spalte $Anzahl
19 }
20
21 ttk::frame .fr
22 ttk::frame .fr2
23 ttk::frame .fr3
24
25 table .fr.table \
26   -rows 20 \
27   -cols 10 \
28   -titlerows 1 \
29   -titlecols 1 \
30   -height 15 \
31   -width 25 \
32   -rowheight 1 \
33   -colwidth 9 \
34   -maxheight 400 \
35   -maxwidth 600 \
36   -selectmode extended \
37   -variable Wert \
38   -xscrollcommand {.fr.xscroll set} \
39   -yscrollcommand {.fr.yscroll set}
40
41 ttk::scrollbar .fr.xscroll -command {.fr.table xview} ↩
     -orient horizontal
42 ttk::scrollbar .fr.yscroll -command {.fr.table yview}
43
44 ttk::button .fr2.btZeileEinfuegenOben -text "Neue Zeile ↩
     oben" -command {ZeileEinfuegen .fr.table 1 -1}
45 ttk::button .fr2.btZeileEinfuegenUnten -text "Neue Zeile ↩
     unten" -command {ZeileEinfuegen .fr.table 3 1}
46 ttk::button .fr2.btSpalteEinfuegenLinks -text "Neue Spalte↩
      links" -command {SpalteEinfuegen .fr.table 1 -1}
47 ttk::button .fr2.btSpalteEinfuegenRechts -text "Neue ↩
     Spalte rechts" -command {SpalteEinfuegen .fr.table 3 1}
48
49 ttk::button .fr3.btZeileLoeschen -text "Zeile loeschen" ↩
     -command {ZeileLoeschen .fr.table 1 1}
50 ttk::button .fr3.btSpalteLoeschen -text "Spalte loeschen" ↩
     -command {SpalteLoeschen .fr.table 1 1}
51
52 pack .fr -fill both -expand 1
```

```
53 pack .fr.xscroll -side bottom -fill x
54 pack .fr.yscroll -side right -fill y
55 pack .fr.table -side right -fill both -expand 1
56
57 pack .fr2 -anchor nw
58 pack .fr2.btZeileEinfuegenOben -side left
59 pack .fr2.btZeileEinfuegenUnten -side left
60 pack .fr2.btSpalteEinfuegenLinks -side left
61 pack .fr2.btSpalteEinfuegenRechts -side left
62
63 pack .fr3 -anchor nw
64 pack .fr3.btZeileLoeschen -side left
65 pack .fr3.btSpalteLoeschen -side left
66
67 for {set Zeile 0} {$Zeile <= 19} {incr Zeile} {
68   for {set Spalte 0} {$Spalte <= 9} {incr Spalte} {
69     if {$Spalte == 0} {
70       if {$Zeile > 0} {
71         set Wert($Zeile,$Spalte) Zeile$Zeile
72       }
73     } else {
74       if {$Zeile == 0} {
75         set Wert($Zeile,$Spalte) Spalte$Spalte
76       } else {
77         set Wert($Zeile,$Spalte) $Zeile$Spalte
78       }
79     }
80   }
81 }
```

Beispiel_335.tcl

	Spalte1	Spalte2	Spalte3	Spalte4	Spalte5	Spalte6	Spalte7
Zeile1	11	12	13	14	15	16	17
Zeile2	21	22	23	24	25	26	27
Zeile3	31	32	33	34	35	36	37
Zeile4	41	42	43	44	45	46	47
Zeile5	51	52	53	54	55	56	57
Zeile6	61	62	63	64	65	66	67
Zeile7	71	72	73	74	75	76	77
Zeile8	81	82	83	84	85	86	87
Zeile9	91	92	93	94	95	96	97
Zeile10	101	102	103	104	105	106	107
Zeile11	111	112	113	114	115	116	117
Zeile12	121	122	123	124	125	126	127
Zeile13	131	132	133	134	135	136	137
Zeile14	141	142	143	144	145	146	147

Neue Zeile oben | Neue Zeile unten | Neue Spalte links | Neue Spalte rechts
Zeile loeschen | Spalte loeschen

Wenn man einige Zeilen und Spalten eingefügt hat:

Beispiel_335.tcl

			Spalte1		Spalte2	Spalte3	Spalte4
Zeile1			11		12	13	14
Zeile2			21		22	23	24
Zeile3			31		32	33	34
Zeile4			41		42	43	44
Zeile5			51		52	53	54
Zeile6			61		62	63	64
Zeile7			71		72	73	74
Zeile8			81		82	83	84
Zeile9			91		92	93	94
Zeile10			101		102	103	104
Zeile11			111		112	113	114
Zeile12			121		122	123	124

Neue Zeile oben | Neue Zeile unten | Neue Spalte links | Neue Spalte rechts
Zeile loeschen | Spalte loeschen

In den Zeilen 5 bis 19 werden die Prozeduren zum Einfügen und Löschen von Zeilen und Spalten definiert. In den Zeilen 44 bis 50 werden die Prozeduren mit den Buttons verknüpft. Die Übergabe der Zeile bzw. Spalte und die Anzahl ist in diesem Beispiel statisch. Dadurch bleibt das Beispiel übersichtlich.

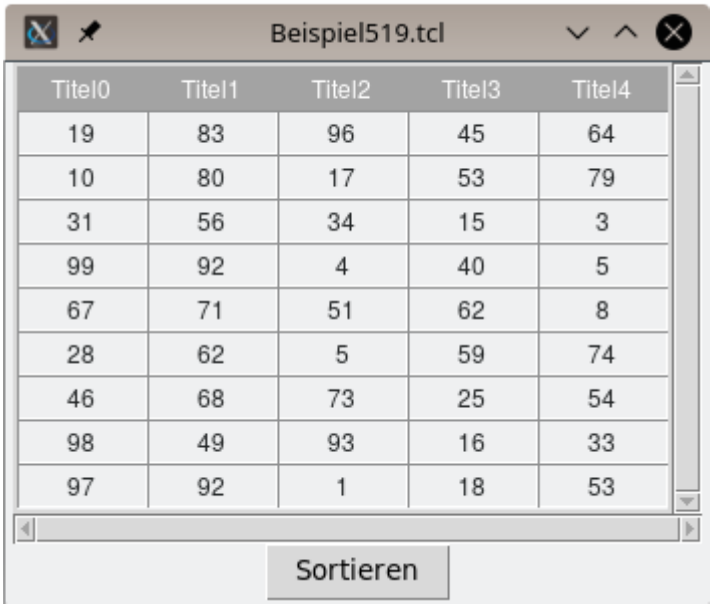
50.3 Tabelle sortieren

Das `TkTable`-Element verfügt über keinen Befehl, die Tabelle zu sortieren. Um dennoch die Einträge zu sortieren, kann man den Inhalt der mit dem `TkTable`-Element verknüpften Array-Variablen in eine Liste überführen und anschließend die Liste sortieren. Die sortierte Liste wird dann wieder in die Array-Variable übernommen.

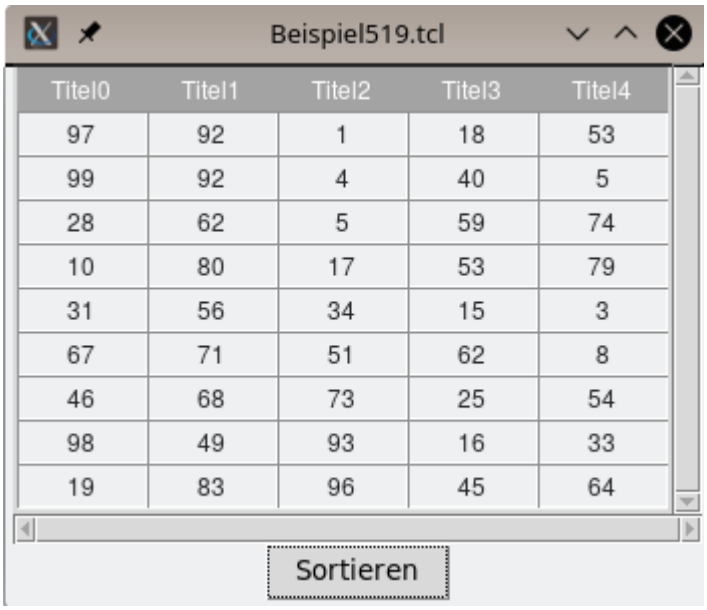
Listing 50.9: Tabelle sortieren (Beispiel519.tcl)

```
1  #!/usr/bin/env wish
2
3  package require Tktable
4
5  proc Sortieren {tmpVar Zeilen Spalten SortierSpalte} {
6    upvar $tmpVar Tabelleninhalt
7
8    set Liste {}
9    for {set i 1} {$i < $Zeilen} {incr i} {
10     set Zeilenwerte {}
11     for {set j 0} {$j < $Spalten} {incr j} {
12       lappend Zeilenwerte $Tabelleninhalt($i,$j)
13     }
14     lappend Liste $Zeilenwerte
15   }
16
17   set Liste [lsort -integer -increasing -index ↩
       $SortierSpalte $Liste]
18
19   for {set i 1} {$i < $Zeilen} {incr i} {
20     set k [expr $i - 1]
21     for {set j 0} {$j < $Spalten} {incr j} {
22       set Tabelleninhalt($i,$j) [lindex [lindex $Liste $k]↩
           $j]
23     }
24   }
25 }
26
27 ttk::frame .fr
28
29 table .fr.table \
30   -rows 10 \
31   -cols 5 \
32   -titlerows 1 \
33   -titlecols 0 \
34   -height 5 \
35   -width 25 \
36   -rowheight 1 \
```

```
37   -colwidth 9 \
38   -maxheight 100 \
39   -maxwidth 400 \
40   -selectmode extended \
41   -variable Tabelleninhalt \
42   -xscrollcommand {.fr.xscroll set} \
43   -yscrollcommand {.fr.yscroll set}
44
45 ttk::scrollbar .fr.xscroll -command {.fr.table xview} ↩
     -orient horizontal
46 ttk::scrollbar .fr.yscroll -command {.fr.table yview}
47
48 ttk::button .btSortieren -text "Sortieren" -command {↩
     Sortieren Tabelleninhalt [.fr.table cget -rows] [↩
     .fr.table cget -cols] 2}
49
50 pack .fr -fill both -expand 1
51 pack .fr.xscroll -side bottom -fill x
52 pack .fr.yscroll -side right -fill y
53 pack .fr.table -side right -fill both -expand 1
54 pack .btSortieren -side bottom
55
56 for {set Zeile 0} {$Zeile <= 9} {incr Zeile} {
57   for {set Spalte 0} {$Spalte <= 4} {incr Spalte} {
58     if {$Zeile == 0} {
59       set Tabelleninhalt($Zeile,$Spalte) Titel$Spalte
60     } else {
61       set Tabelleninhalt($Zeile,$Spalte) [expr int(100↩
           *rand())]
62     }
63   }
64 }
```



Nach einem Klick auf den `Sortieren`-Button, wird die Tabelle nach der dritten Spalte sortiert:



In Zeile 41 wird das `TkTable`-Element mit der Array-Variablen `Tabelleninhalt` verknüpft. In den Zeilen 56 bis 64 wird die Variable (und somit die Tabelle) mit Zufallszahlen befüllt. Die Zeilen 5 bis 25 umfassen die `Sortieren`-Prozedur. In den Zeilen 8 bis 15 wird der Inhalte der Variablen `Tabelleninhalt` in eine Liste überführt. In Zeile 9 wird die Tabellen-Variable zeilenweise durchlaufen. Dabei wird die Zeile mit dem Index 0 weggelassen, weil es sich um die Titelzeile handelt. In Zeile 11 werden die Spalten der Tabellen-Variablen durchlaufen. Jeder Eintrag in einer Zeile wird zu einer Liste (`Zeilenwerte`) zusammengefasst (Zeile 12). In 14 Zeile werden die einzelnen Liste mit den Zeilenwerten in einer Gesamtliste gespeichert. Somit handelt es sich um Listen in einer Liste. In Zeile 17 wird die Gesamtliste sortiert. In den Zeilen 19 bis 24 wird die sortierte Liste wieder in die Tabellen-Variable zurückübertragen.

51 Canvas (Zeichenfläche)

51.1 Zeichenfläche und Objekte

Die Zeichenfläche heißt Canvas. Man kann sie wie jedes andere Element frei in einem Dialog platzieren. In die Zeichenfläche kann man geometrische Figuren wie Linien, Rechtecke, Kreise usw. aber auch Polygone, Texte und Bilder/Bitmaps setzen. Die Objekte können nachträglich verändert, gelöscht und bewegt werden. Somit kann man mit Hilfe der Zeichenfläche Animationen erstellen.

Außerdem kann man ein Canvas als Container für andere GUI-Elemente (Label, Entry, Button usw.) verwenden, wenn die GUI größer als das Programmfenster (bzw. größer als die Bildschirmfläche) ist.

Die Ecke links oben hat die Koordinaten (0/0). Text kann auch gedreht ausgegeben werden. Ein Rechteck kann man nicht drehen. Hier muss man sich mit einem Polygon behelfen (siehe Beispiel 308).

Tabelle 51.1: Die wichtigsten Optionen

Option	Beschreibung
`-width Pixel`	Breite
`-height Pixel`	Höhe
`-background Farbe`	Hintergrundfarbe, wenn man einen Farbname (z. B. `blue`) verwendet
`-background #F0E852`	Hintergrundfarbe, wenn man Hexadezimalwerte verwendet
`-background [format "#%02x%02x%02x" 249 208 10]`	Hintergrundfarbe, wenn man dezimale RGB-Werte verwendet

Tabelle 51.2: Die wichtigsten Aktionen

Aktion	Beschreibung
`canvas .c -width 250 -height 60`	erstellt eine leere Zeichenfläche
`set Elementname [.c create Element]`	Erzeugt ein Objekt
`.c delete Elementname`	Löscht ein Objekt
`.c moveto Elementname x y`	Setzt ein Objekt auf Position x/y

Tabelle 51.2: Die wichtigsten Aktionen

Aktion	Beschreibung
`.c move Elementname x y`	Verschiebt ein Objekt in x/y-Richtung
`.c raise Elementname`	Setzt ein Objekt in den Vordergrund
`.c raise Elementname AndererElementname`	Setzt ein Objekt vor ein anderes Objekt
`.c lower Elementname`	Setzt ein Objekt in den Hintergrund
`.c lower Elementname AndererElementname`	Setzt ein Objekt hinter ein anderes Objekt
`.c itemconfigure Elementname -fill red`	Ändert die Eigenschaft eines Objekts
`[.c bbox Elementname]`	Ermittelt die Eckpunkte eines Rahmens, der genau um das Objekte passt
`[.c bbox Elementname Elementname Elementname]`	Ermittelt die Eckpunkte eines Rahmens, der genau um die Objekte passt

Tabelle 51.3: Geometrische Formen

Form	Beschreibung
`arc`	Kreissegment (Kuchenstück)
`bitmap`	Bitmap-Grafik
`image`	Bild/Foto
`line`	Linie oder Punkt
`oval`	Oval oder Kreis
`polygon`	Polygon (Vieleck)
`rectangle`	Rechteck oder Quadrat
`text`	Text
`window`	Fenster, in das weitere Elemente eingefügt werden können

Tabelle 51.4: Die wichtigsten line-Optionen

Line-Option	Beschreibung
`-width Pixel`	Breite der Linie
`-fill Farbe`	Farbe der Linie
`-stipple Füllmuster`	Füllmuster. Wird nur ausgeführt, wenn auch die Option `-fill` gesetzt ist.
`-dash Linienmuster`	Linienmuster

Tabelle 51.5: Die wichtigsten oval-Optionen

Oval-Option	Beschreibung
`-width Pixel`	Breite der Umrandung
`-outline Farbe`	Farbe der Umrandung
`-fill Farbe`	Füllfarbe
`-stipple Füllmuster`	Füllmuster. Wird nur ausgeführt, wenn auch die Option `-fill` gesetzt ist.
`-dash Linienmuster`	Linienmuster

Tabelle 51.6: Die wichtigsten rectangle-Optionen

Rectangle-Option	Beschreibung
`-width Pixel`	Breite der Umrandung
`-outline Farbe`	Farbe der Umrandung
`-fill Farbe`	Füllfarbe
`-stipple Füllmuster`	Füllmuster. Wird nur ausgeführt, wenn auch die Option `-fill` gesetzt ist.
`-dash Linienmuster`	Linienmuster

Tabelle 51.7: Die wichtigsten arc-Optionen

Arc-Option	Beschreibung
`-start Grad`	Start des Kreissegments. Angabe in Grad. Die Horizontale hat 0 Grad. Die Drehrichtung ist entgegen dem Uhrzeigersinn.
`-extent`	Winkelbreite des Kreissegments
`-width Pixel`	Breite der Umrandung
`-outline Farbe`	Farbe der Umrandung
`-fill Farbe`	Füllfarbe
`-stipple Füllmuster`	Füllmuster. Wird nur ausgeführt, wenn auch die Option `-fill` gesetzt ist.
`-dash Linienmuster`	Linienmuster

Tabelle 51.8: Die wichtigsten polygon-Optionen

Polygon-Option	Beschreibung
`-width Pixel`	Breite der Umrandung
`-outline Farbe`	Farbe der Umrandung
`-fill Farbe`	Füllfarbe
`-stipple Füllmuster`	Füllmuster. Wird nur ausgeführt, wenn auch die Option `-fill` gesetzt ist.
`-dash Linienmuster`	Linienmuster

Tabelle 51.9: Die wichtigsten text-Optionen

Text-Option	Beschreibung
`-text "Text"`	Text
`-fill Farbe`	Textfarbe
`-width Zeilenlänge`	Zeilenlänge
`-angle Grad`	Drehung des Textes in Grad

Tabelle 51.9: Die wichtigsten text-Optionen

Text-Option	Beschreibung

`-anchor n`
`-anchor ne`
`-anchor e`
`-anchor se`
`-anchor s`
`-anchor sw`
`-anchor w`
`-anchor nw`
`-anchor center`
	Verankerung der Textbox gemäß der Himmelsrichtungen:
	`n`=North (Norden)
	`e`=East (Osten)
	`s`=South (Süden)
	`w`=West (West)
	`center`=zentriert

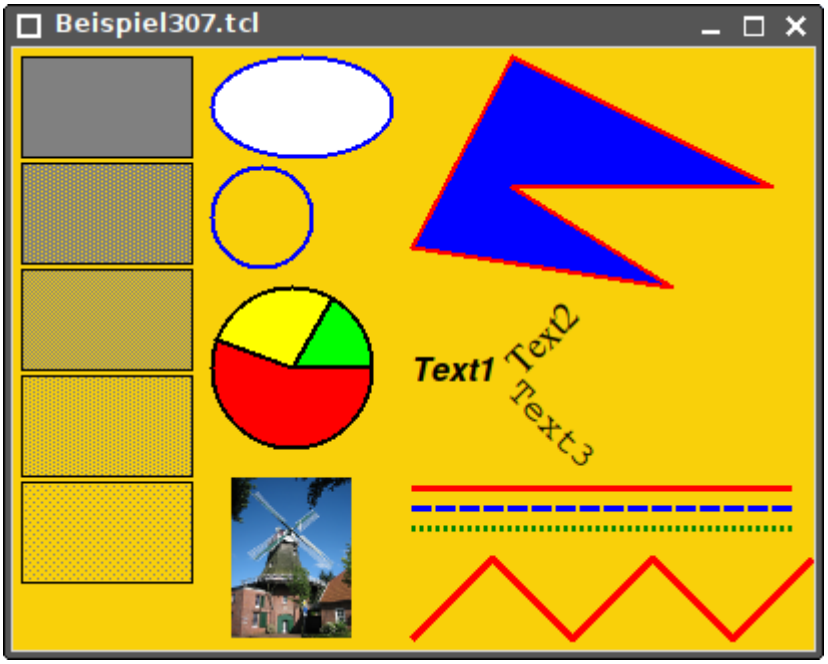
`-justify left`
`-justify right`
`-justify center`
	Ausrichtung des Textes: linksbündig, rechtsbündig, zentriert. Die Option gilt nur für mehrzeiligen Text.

`-font "Schriftart Größe Stil"`	Schriftart, -größe und -stil (fett, kursiv)

Listing 51.1: Zeichenfläche mit einfachen geometrischen Objekten (Beispiel307.tcl)

```
#!/usr/bin/env wish

canvas .c -width 400 -height 300 -background [format "#%02x%02x%02x" 249 208 10]
pack .c

set Rechteck1 [.c create rectangle 5 5 90 55 -width 1 -outline black -fill grey]
set Rechteck2 [.c create rectangle 5 58 90 108 -width 1 -outline black -fill grey -stipple "gray75"]
set Rechteck3 [.c create rectangle 5 111 90 161 -width 1 -outline black -fill grey -stipple "gray50"]
set Rechteck4 [.c create rectangle 5 164 90 214 -width 1 -outline black -fill grey -stipple "gray25"]
set Rechteck5 [.c create rectangle 5 217 90 267 -width 1 -outline black -fill grey -stipple "gray12"]

set Linie1 [.c create line 200 220 390 220 -width 3 -fill red]
set Linie2 [.c create line 200 230 390 230 -width 3 -fill blue -dash {10 2}]
set Linie3 [.c create line 200 240 390 240 -width 3 -fill green -dash {2 2}]
set Linie4 [.c create line 200 295 240 255 280 295 320 255 360 295 400 255 -width 3 -fill red]

set Oval [.c create oval 100 5 190 55 -width 2 -outline blue -fill white]
```

```
18 set Kreis [.c create oval 100 60 150 110 -width 2 -outline blue -fill ""]
19 set Kreissegment1 [.c create arc 100 120 180 200 -start 0 -extent 60 -width 2 -outline black -fill [format "#%02x%02x%02x" 0 255 0]]
20 set Kreissegment2 [.c create arc 100 120 180 200 -start 60 -extent 100 -width 2 -outline black -fill [format "#%02x%02x%02x" 255 255 0]]
21 set Kreissegment3 [.c create arc 100 120 180 200 -start 160 -extent 200 -width 2 -outline black -fill [format "#%02x%02x%02x" 255 0 0]]
22
23 set ListePunkte {}
24 set ListePunkte [list 200 100 250 5 380 70 250 70 330 120 200 100]
25 set Polygon [.c create polygon $ListePunkte -width 2 -outline red -fill blue]
26
27 set Text1 [.c create text 200 160 -text "Text1" -anchor w -font "Helvetica 12 bold italic"]
28 set Text2 [.c create text 250 160 -text "Text2" -anchor w -angle 45 -font "Times 14"]
29 set Text3 [.c create text 250 170 -text "Text3" -anchor w -angle -45 -font "Courier 14"]
30
31 set Foto2 [image create photo Foto -file "foto.ppm"]
32 .c create image 110 215 -anchor nw -image Foto
```



In Zeile 3 wird die Zeichenfläche definiert. Die Hintergrundfarbe wird als RGB-Farbwert angegeben.

Ab Zeile 6 werden mit dem `create`-Befehl verschiedene Objekte gezeichnet. Der Befehl hat als Rückgabewert einen Verweis auf das Objekt, so dass es später im Programm angesprochen werden kann (z. B. um die Objekteigenschaften zu ändern, das Objekt zu verschieben oder zu löschen). Dieser Verweis auf das Objekt wird in Variablen gespeichert.
In den Zeilen 6 bis 10 werden fünf Rechtecke erstellt, die mit verschiedenen Mustern gefüllt sind. Das Muster wird mit der Option `-stipple` festgelegt. Wenn die Option `-stipple` weggelassen wird, wird die Form vollständig gefüllt. Die Option wird allerdings nur dann beachtet, wenn mit der Option `-fill` auch eine Füllfarbe festgelegt wird.
Die Koordinaten der Rechtecke sind immer x/y-Wertepaare. Das erste Wertepaar bezeichnet die Ecke links oben des Rechtecks, das zweite Wertepaar die Ecke rechts unten. In Zeile 12 wird eine Linie gezeichnet. Das erste Koordinatenpaar legt den Startpunkt der Linie fest, das zweite Paar den Endpunkt.
In den Zeilen 13 und 14 werden zwei weitere Linien gezeichnet. Mit der Option `-dash` kann man das Linienmuster festlegen. Die erste Zahl in der Klammer gibt an, wie viele Bildpunkte der Linie gezeichnet werden sollen, die zweite Zahl in der Klammer gibt an, wie viele Bildpunkte ausgelassen werden sollen.
In Zeile 15 wird eine Zick-Zack-Linie gezeichnet. Hierzu werden die x-/y-Koordinate der Reihe nach aufgelistet.
In Zeile 17 wird ein Oval gezeichnet und in Zeile 18 ein Kreis. Beides mal wird der Befehl `oval` verwendet. Ob ein Oval oder ein Kreis entsteht, hängt nur von den Koordinatenpaaren ab.
In den Zeilen 19 bis 21 werden drei Kreissegmente gezeichnet. Die Koordinatenpaare definieren den (virtuellen) Vollkreis, in den das gewünschte Kreissegment gezeichnet wird. Die Option `-start` legt fest, in welchem Winkel zur Horizontale das Kreissegment beginnt. Positive Winkelangaben werden immer gegen den Uhrzeigersinn abgetragen. Die Option `-extent` gibt die Winkelbreite des Kreissegments an. Somit startet das `Kreissegment1` bei 0 Grad und geht (entgegen dem Uhrzeigersinn) bis 60 Grad. Das `Kreissegment2` startet bei 60 Grad und geht bis 160 Grad (hat somit eine Winkelbreite von 100 Grad). Das `Kreissegment3` startet bei 160 Grad und endet bei 360 Grad (Winkelbreite 200 Grad). Die Füllfarbe ist in dem Beispiel als RGB-Farbwert angegeben.
In den Zeilen 23 bis 25 wird ein Polygon erstellt. Ein Polygon ist eine geschlossene geometrische Figur mit beliebigem Umriss. Der Umriss wird durch Koordinatenpaare definiert, die als Liste an den `polygon`-Befehl übergeben werden.
In den Zeilen 27 bis 29 werden drei Texte ausgegeben. Die Texte können mit der Option `-angle` gedreht werden.
In den Zeilen 31 und 32 wird ein Bild platziert. Die Option `-anchor` gibt an, wo das Bild verankert werden soll. Die Option `-image` enthält den Verweis auf das Bild.

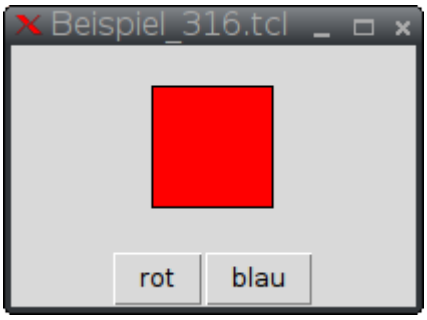
51.2 Objekteigenschaften ändern

Man kann die gezeichneten Objekte nachträglich ändern, löschen oder bewegen. Dazu speichert man beim Erstellen des Objekts einen Verweis auf das Objekt in einer Variablen, so dass man später auf das Objekt zugreifen kann.

Listing 51.2: Objekteigenschaften nachträglich ändern (Beispiel316.tcl)

```
1 #!/usr/bin/env wish
2
```

```
3  canvas .c -width 200 -height 100
4  set Rechteck [.c create rectangle 70 20 130 80 -width 1 ↩
       -outline black -fill grey]
5  ttk::frame .fr
6  ttk::button .fr.btRot -text "rot" -command {.c ↩
       itemconfigure $Rechteck -fill red}
7  ttk::button .fr.btBlau -text "blau" -command {.c ↩
       itemconfigure $Rechteck -fill blue}
8  pack .c
9  pack .fr
10 pack .fr.btRot -side left
11 pack .fr.btBlau -side left
```



In den Zeilen 6 und 7 wird die Farbe des Quadrats geändert.

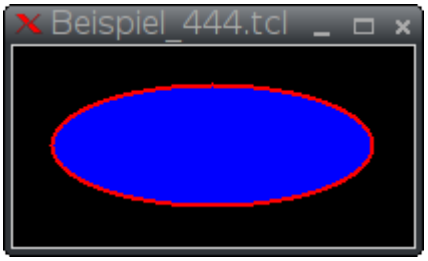
Auch wenn die Maus- und Tastaturereignisse erst in Kapital 44 auf Seite 533 behandelt werden, soll schon hier gezeigt werden, dass man ein Element der Zeichenfläche mit einem Maus-Ereignis koppeln kann.

Listing 51.3: Mausklick auf ein Element (Beispiel444.tcl)

```
1  #!/usr/bin/env wish
2
3  proc MausKlick {Element} {
4          .c itemconfigure $Element -fill blue
5          update idletasks
6  }
7
8  proc MausRelease {Element} {
9          .c itemconfigure $Element -fill red
10         update idletasks
11 }
12
13 canvas .c -width 200 -height 100 -background black
14 set Oval [.c create oval 20 20 180 80 -width 2 -outline ↩
       red -fill red]
15 .c bind $Oval <ButtonPress-1> {MausKlick $Oval}
16 .c bind $Oval <ButtonRelease-1> {MausRelease $Oval}
17 pack .c
```

Wenn das Programm startet:

Wenn sich die Maus im Oval befindet und die linke Maustaste gedrückt wird:



Nach dem Loslassen der Maustaste:

Die Maus- und Tastaturereignisse werden innerhalb der Zeichenfläche `.c` definiert (Zeilen 15 und 16). In Zeile 15 wird das Oval durch den Befehl `bind` mit dem Klick auf die linke Maustaste verknüpft. In Zeile 16 wird das Oval mit dem Loslassen der linken Maustaste verknüpft.

Listing 51.4: Koordinaten nachträglich ändern (Beispiel318.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Oben {Linie} {
4    .c coords $Linie {10 100 190 20}
5  }
6
7  proc Unten {Linie} {
8    set Wert 180
9    set Koordinaten [list 10 100 190 $Wert]
10   .c coords $Linie $Koordinaten
11 }
12
13 canvas .c -width 200 -height 200
14 pack .c
15
```

```
16 set Linie [.c create line 10 100 190 100 -width 3 -fill red]
17
18 ttk::frame .fr
19 ttk::button .fr.btOben -text "oben" -command {Oben $Linie}
20 ttk::button .fr.btUnten -text "unten" -command {Unten $Linie}
21 pack .fr
22 pack .fr.btOben -side left
23 pack .fr.btUnten -side left
```

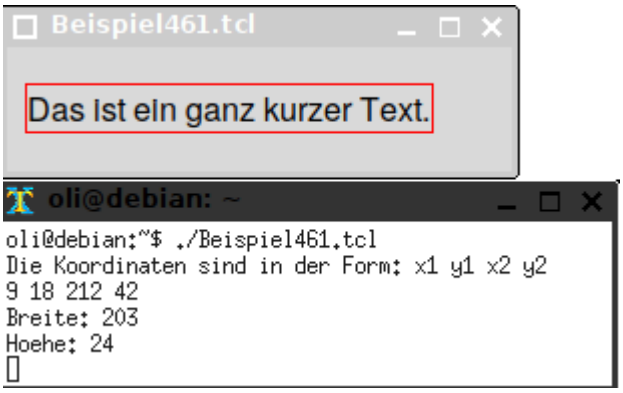
In den Zeilen 20 und 21 werden die beiden Prozeduren `Oben` bzw. `Unten` aufgerufen. Die Zeilen 3 bis 5 umfassen die Prozedur `Oben`. In Zeile 4 werden neue Koordinaten für die Linie gesetzt. Die Zeilen 7 bis 11 umfassen die Prozedur `Unten`. Hier werden die Koordinaten unter Verwendung von Variablen gesetzt. In Zeile 8 wird eine Koordinate in der Variablen `Wert` gespeichert. In Zeile 9 werden die neuen Koordinaten in der Variablen `Koordinaten` abgelegt. Dabei handelt es sich um eine Liste. In Zeile 10 werden die neuen Koordinaten für die Linie gesetzt.
Bei vielen Canvas-Elementen (z. B. `rectangle`, `oval`) gibt man die Größe bereits beim Erzeugen des Elements an. Bei dem `text`-Element hängt die Höhe und Breite von der gewählten Schriftart und -größe ab. Um dennoch die Breite und Höhe ermitteln zu können, verwendet man den Befehl `bbox`. Der Befehl gibt die Koordinaten eines Rahmens zurück, der genau um das `text`-Element passt.

51.3 Breite und Höhe eines Objekts

Listing 51.5: Breite und Höhe eines Objekts ermitteln (Beispiel461.tcl)

```
1 #!/usr/bin/env wish
2
3 canvas .c -width 250 -height 60
4 set Text [.c create text 10 30 -text "Das ist ein ganz kurzer Text." -font "helvetica 12" -anchor w]
```

```
5  pack .c
6
7  set Ecken [.c bbox $Text]
8  .c create rectangle $Ecken -width 1 -outline red
9
10 puts "Die Koordinaten sind in der Form: x1 y1 x2 y2"
11 puts $Ecken
12
13 lassign $Ecken x1 y1 x2 y2
14
15 set Breite [expr $x2 - $x1]
16 set Hoehe [expr $y2 -$y1]
17 puts "Breite: $Breite"
18 puts "Hoehe: $Hoehe"
```

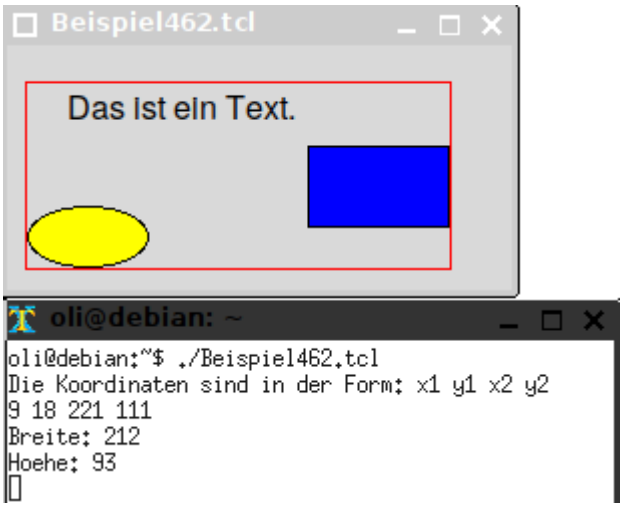


In Zeile 4 wird ein Text erzeugt. In Zeile 7 werden die Eckpunkte des Rahmens, der genau um den Text passt, abgefragt. Es handelt sich dabei um eine Liste mit den Koordinaten in der Reihenfolge x1, y1, x2, y2. In Zeile 8 wird zur Veranschaulichung der Rahmen gezeichnet. In Zeile 13 werden die Eckpunkte den Variablen `x1`, `y1`, `x2` und `y2` zugeordnet. In den Zeilen 15 bis 18 werden die Breite und Höhe des Rahmens ausgerechnet und angezeigt. Der `bbox`-Befehl kann auch auf mehrere Objekte angewendet werden.

Listing 51.6: Breite und Höhe mehrerer Objekte ermitteln (Beispiel462.tcl)

```
1  #!/usr/bin/env wish
2
3  canvas .c -width 250 -height 120
4  set Text [.c create text 30 30 -text "Das ist ein Text." ↩
     -font "helvetica 12" -anchor w]
5  set Oval [.c create oval 10 80 70 110 -fill yellow]
6  set Rechteck [.c create rectangle 150 50 220 90 -fill blue↩
     ]
7  pack .c
8
9  set Ecken [.c bbox $Text $Oval $Rechteck]
10 .c create rectangle $Ecken -width 1 -outline red
11
12 puts "Die Koordinaten sind in der Form: x1 y1 x2 y2"
13 puts $Ecken
```

```
14
15 lassign $Ecken x1 y1 x2 y2
16
17 set Breite [expr $x2 - $x1]
18 set Hoehe [expr $y2 -$y1]
19 puts "Breite: $Breite"
20 puts "Hoehe: $Hoehe"
```



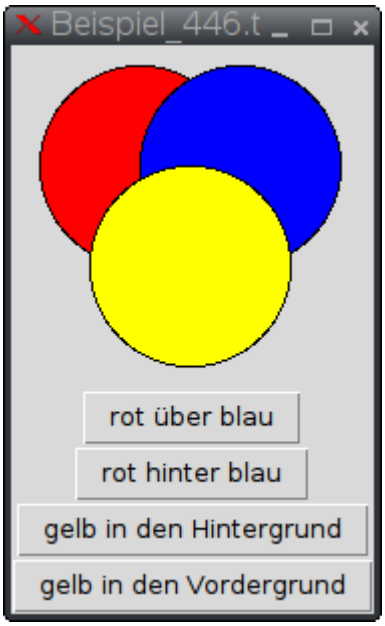
In den Zeilen 4 bis 6 werden drei Objekte ermittelt. In Zeile 9 werden Eckpunkte des Rahmens ermittelt, der genau um die drei Objekte passt. In Zeile 10 wird zur Veranschaulichung der Rahmen gezeichnet. In Zeile 15 werden die Eckpunkte den Variablen `x1`, `y1`, `x2` und `y2` zugeordnet. In den Zeilen 17 bis 20 werden die Breite und Höhe des Rahmens ausgerechnet und angezeigt.

51.4 Objekt in den Vorder-/Hintergrund setzen

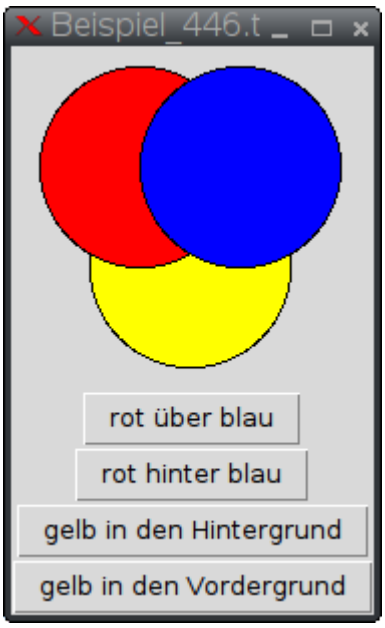
Listing 51.7: Objekt in den Vordergrund bzw. Hintergrund setzen (Beispiel446.tcl)

```
1 #!/usr/bin/env wish
2
3 canvas .c -width 170 -height 170
4 set KreisRot [.c create oval 10 10 110 110 -width 1 ↩
    -outline black -fill red]
5 set KreisBlau [.c create oval 60 10 160 110 -width 1 ↩
    -outline black -fill blue]
6 set KreisGelb [.c create oval 35 60 135 160 -width 1 ↩
    -outline black -fill yellow]
7
8 ttk::frame .fr
9 ttk::button .fr.bt1 -text "rot ueber blau" -command {.c ↩
    raise $KreisRot $KreisBlau}
10 ttk::button .fr.bt2 -text "rot hinter blau" -command {.c ↩
    lower $KreisRot $KreisBlau}
```

```
11 ttk::button .fr.bt3 -text "gelb in den Hintergrund" ↲
      -command {.c lower $KreisGelb}
12 ttk::button .fr.bt4 -text "gelb in den Vordergrund" ↲
      -command {.c raise $KreisGelb}
13 grid .fr.bt1 -row 0 -column 0
14 grid .fr.bt2 -row 1 -column 0
15 grid .fr.bt3 -row 2 -column 0
16 grid .fr.bt4 -row 3 -column 0
17
18 pack .c
19 pack .fr
```



Nach einem Klick auf den Button `gelb in den Hintergrund`:



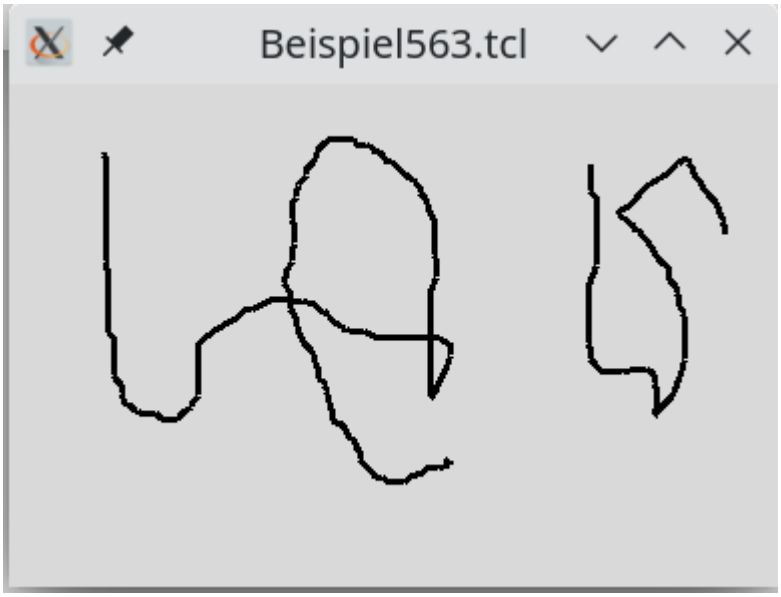
Wenn das Programm gestartet wird, werden die Kreise in der Reihenfolge rot, blau, gelb dargestellt. Der rote Kreis wird somit vom blauen Kreis überlagert, und der gelbe Kreis überlagert die beiden anderen Kreise. Mit den Befehlen `raise` und `lower` kann man die Kreise in den Vordergrund / Hintergrund setzen oder vor / hinter einen anderen Kreis. In Zeile 9 wird der rote Kreis vor den blauen Kreis gesetzt. In Zeile 10 wird der rote Kreis hinter den blauen Kreis gesetzt. In Zeile 11 wird der gelbe Kreis in den Hintergrund (d. h. hinter alle anderen Objekte) gesetzt. In Zeile 12 wird der gelbe Kreis in den Vordergrund (d. h. vor alle anderen Objekte) gesetzt.

51.5 Mit der Maus zeichnen

Listing 51.8: Mit der Maus zeichnen (Beispiel563.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Zeichnen {x y} {
4          global xAlt
5          global yAlt
6          if {$xAlt ne ""} {
7                  set Linie [.c create line $xAlt $yAlt $x ↩
                      $y -width 3 -fill black]
8          }
9          set xAlt $x
10         set yAlt $y
11 }
12
13 proc MausAn {x y} {
14         bind .c <Motion> {Zeichnen [%W canvasx %x] [%W ↩
              canvasy %y]}
15 }
16
```

```
17 proc MausAus {} {
18         global xAlt
19         global yAlt
20         set xAlt ""
21         set yAlt ""
22         bind .c <Motion> {}
23 }
24
25 set xAlt ""
26 set yAlt ""
27 canvas .c -width 200 -height 200
28 pack .c -fill both -expand yes
29 bind .c <ButtonPress-1> {MausAn [%W canvasx %x] [%W ↩
   canvasy %y]}
30 bind .c <ButtonRelease-1> {MausAus}
```

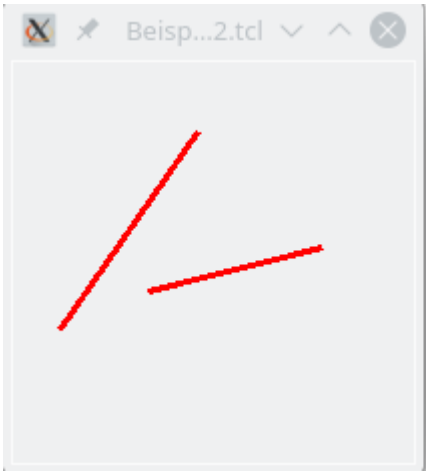


Das Programm zeichnet eine schwarze Linie, wenn man den Mauszeiger über die Zeichenfläche bewegt. In den Zeilen 25 und 26 werden die Startkoordinaten der Linie geleert. In den Zeilen 27 und 28 wird die Zeichenfläche erstellt. In Zeile 29 wird die Zeichenfläche mit dem Mausereignis `Klick auf die linke Taste` verknüpft. Die Prozedur `MausAn` verknüpft die Zeichenfläche mit dem Mausereignis `Maus bewegen`. Wenn die Maus (bei gedrückter linker Taste) bewegt wird, wird die Prozedur `Zeichnen` aufgerufen. Die Prozedur `Zeichnen` zeichnet eine Linie von der letzten Mausposition (gespeichert in den Variablen `xAlt` und `yAlt`) zur aktuellen Mausposition. In Zeile 30 wird die Zeichenfläche mit dem Mausereignis `Linke Taste loslassen` verknüpft. Die Prozedur `MausAus` löscht das Mausereignis `Maus bewegen`.

Ein weiteres Beispiel: Wenn man mit der linken Maustaste in die Zeichenfläche klickt, wird der Startpunkt einer Linie gesetzt. Bewegt man die Maus, wird der Endpunkt der Linie verschoben. Mit einem erneuten Klick auf die linke Maustaste wird die Linie beendet. Ein Klick auf die rechte Maustaste bricht das Zeichnen der Linie ab.

Listing 51.9: Mit der Maus zeichnen (weiteres Beispiel) (Beispiel512.tcl)

```
#!/usr/bin/env wish

proc MausBewegungAn {x y} {
        global ObjektID

        set ObjektID [.fr.c create line $x $y $x $y -width 3 -fill red]
        bind .fr.c <Motion> {.fr.c coords $ObjektID [lindex [.fr.c coords $ObjektID] 0] [lindex [.fr.c coords $ObjektID] 1] [%W canvasx %x] [%W canvasy %y]}
        bind .fr.c <ButtonPress-1> {MausBewegungAus}
        bind .fr.c <ButtonPress-3> {MausBewegungAbbruch}
}
proc MausBewegungAus {} {
        bind .fr.c <Motion> {}
        bind .fr.c <ButtonPress-3> {}
        bind .fr.c <ButtonPress-1> {MausBewegungAn [%W canvasx %x] [%W canvasy %y]}
}

proc MausBewegungAbbruch {} {
        global ObjektID

        .fr.c delete $ObjektID
        bind .fr.c <Motion> {}
        bind .fr.c <ButtonPress-3> {}
        bind .fr.c <ButtonPress-1> {MausBewegungAn [%W canvasx %x] [%W canvasy %y]}
}

ttk::frame .fr
canvas .fr.c -width 200 -height 200
pack .fr.c -expand yes -fill both -side top
pack .fr -expand yes -fill both -side top
bind .fr.c <ButtonPress-1> {MausBewegungAn [%W canvasx %x] [%W canvasy %y]}
```



In den Zeilen 26 bis 29 wird eine Zeichenfläche erzeugt. In Zeile 30 wird die Zeichenfläche mit dem Mausereignis `Klick auf die linke Taste` verknüpft. Wenn die linke Maustaste geklickt wird, wird die Prozedur `MausBewegungAn` aufgerufen. Die Prozedur `MausBewegungAn` hat folgenden Inhalt: In Zeile 6 wird eine Linie erzeugt. In Zeile 7 wird die Bewegung der Maus mit der Linie verknüpft, so dass der Endpunkt der Linie immer auf die Position des Mauszeigers gesetzt wird. In Zeile 8 wird das Mausereignis `Klick auf die linke Taste` neu definiert. Jetzt soll ein Klick auf die linke Maustaste die Bewegung des Linienendpunkts beenden. In Zeile 9 wird das Mausereignis `Klick auf die rechte Taste` festgelegt: ein Klick auf die rechte Maustaste soll die Linie löschen. In der Prozedur `MausBewegungAus` (Zeilen 11 bis 15) werden die Maus-Ereignisse gelöscht (Zeilen 12 und 13) sowie das ursprüngliche Mausereignis wieder hergestellt (Zeile 14). In der Prozedur `MausBewegungAbbruch` werden die Linie (Zeile 20) und die Maus-Ereignisse (Zeilen 21 und 22) gelöscht sowie das ursprüngliche Mausereignis wieder hergestellt (Zeile 23).

Das nächste Beispiel zeigt, wie man ein gezeichnetes Objekt mit der Maus auswählt und bewegen kann. Wenn man mit der linken Maustaste in die Zeichenfläche klickt, wird das nächst gelegene Objekt ausgewählt und kann (bei gedrückter linker Maustaste) bewegt werden.

Listing 51.10: Objekt mit der Maus bewegen (Beispiel564.tcl)

```
1  #!/usr/bin/env wish
2
3  proc LinieZeichnen {x1 y1 x2 y2 Farbe} {
4          set ObjektID [.c create line $x1 $y1 $x2 $y2 ↩
              -width 3 -fill $Farbe]
5  }
6
7  proc MausAn {x y} {
8          global ObjektID
9          global xAlt
10         global yAlt
11         set ObjektID [.c find closest $x $y]
12         bind .c <Motion> {MausBewegen [%W canvasx %x] [%W ↩
              canvasy %y]}
13         set xAlt $x
```

```
14          set yAlt $y
15 }
16
17 proc MausAus {} {
18          bind .c <Motion> {}
19 }
20
21 proc MausBewegen {x y} {
22          global ObjektID
23          global xAlt
24          global yAlt
25          set xVerschiebung [expr $x - $xAlt]
26          set yVerschiebung [expr $y - $yAlt]
27          .c move $ObjektID $xVerschiebung $yVerschiebung
28          set xAlt $x
29          set yAlt $y
30 }
31
32 canvas .c -width 200 -height 200
33 pack .c -expand yes -fill both -side top
34 bind .c <ButtonPress-1> {MausAn [%W canvasx %x] [%W ↩
   canvasy %y]}
35 bind .c <ButtonRelease-1> {MausAus}
36
37 set ObjektID ""
38 set xAlt ""
39 set yAlt ""
40 LinieZeichnen 20 20 80 80 red
41 LinieZeichnen 100 180 170 60 blue
```

In den Zeilen 32 und 33 wird die Zeichenfläche erstellt. In den Zeilen 37 bis 41 werden die Variablen initialisiert und zwei Linien gezeichnet. In Zeile 34 wird die Zeichenfläche mit dem Mausereignis `Klick auf die linke Taste` verknüpft. Die Prozedur `MausAn`

sucht das Objekt, das dem Mauszeiger am nächsten liegt (Zeile 11) und verknüpft die Zeichenfläche mit dem Mausereignis `Maus bewegen`. Außerdem wird die aktuelle Position des Mauszeigers gespeichert (Zeilen 13 und 14). Die Prozedur `MausBewegen` berechnet, wie sich die Position des Mauszeigers verändert hat (Zeilen 25 und 26) und verschiebt in demselben Maß das Objekt (Zeile 27). Die neue Position des Mauszeigers wird wieder gespeichert (Zeilen 28 und 29).

51.6 Objekt bewegen

Listing 51.11: Objekte bewegen (Beispiel317.tcl)

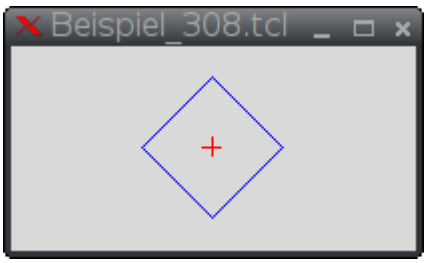
```
1  #!/usr/bin/env wish
2
3  canvas .c -width 500 -height 100
4  set Rechteck [.c create rectangle 220 20 280 80 -width 1 ↩
     -outline black -fill grey]
5  ttk::frame .fr
6  ttk::button .fr.btRechts -text "rechts" -command {.c ↩
     moveto $Rechteck 420 20}
7  ttk::button .fr.btLinks -text "links" -command {.c moveto ↩
     $Rechteck 10 20}
8  ttk::button .fr.btSchrittweiseLinks -text "etwas links" ↩
     -command {.c move $Rechteck -10 0 }
9  ttk::button .fr.btSchrittweiseRechts -text "etwas rechts" ↩
     -command {.c move $Rechteck 10 0}
10 pack .c
11 pack .fr
12 pack .fr.btLinks -side left
13 pack .fr.btRechts -side left
14 pack .fr.btSchrittweiseLinks -side left
15 pack .fr.btSchrittweiseRechts -side left
```

In den Zeilen 6 bis 9 wird das Quadrat bewegt.
Ein Rechteck bzw. Quadrat, das mit dem Befehl `rectangle` erstellt wurde, kann nicht gedreht werden. Stattdessen muss man das Rechteck als Polygon zeichnen und mit einer eigenen Prozedur die Drehung berechnen.

Listing 51.12: Ein Rechteck oder Quadrat drehen (Beispiel308.tcl)

```
1  #!/usr/bin/env wish
```

```
2
3 proc Drehen {Liste Grad DPx DPy} {
4   # Liste = Liste der Eckpunkt des Polygons
5   # Grad = Drehwinkel (0 bis 360 Grad), entgegen dem ↩
      Uhrzeigersinn
6   # DPx = x-Koordinate des Drehpunkts
7   # DPy = y-Koordinate des Drehpunkts
8
9   # Rueckgabe: Liste der transformierten Eckpunkte
10
11  set Pi 3.1415926535897931
12  set Grad2 [expr $Grad/180.0*$Pi]
13
14  set ListeGedreht {}
15  foreach {x y} $Liste {
16    set x2 [expr $x-$DPx]
17    set y2 [expr $y-$DPy]
18    lappend ListeGedreht [expr round($x2 * cos($Grad2) + ↩
        $y2 * sin($Grad2))+$DPx]
19    lappend ListeGedreht [expr round(-$x2 * sin($Grad2) + ↩
        $y2 * cos($Grad2))+$DPy]
20  }
21  return $ListeGedreht
22 }
23
24 canvas .c -width 200 -height 100
25 pack .c
26
27 set Linie1 [.c create line 95 50 105 50 -width 1 -fill red↩
    ] ; # Drehpunkt
28 set Linie2 [.c create line 100 45 100 55 -width 1 -fill ↩
    red] ; # Drehpunkt
29 set ListePunkte {}
30 set ListePunkte [list 75 25 125 25 125 75 75 75] ; # ↩
    Rechteck
31 set Grad 45
32 set ListePunkteGedreht [Drehen $ListePunkte $Grad 100 50]
33 set Polygon [.c create polygon $ListePunkteGedreht -width ↩
    1 -outline blue -fill ""]
```



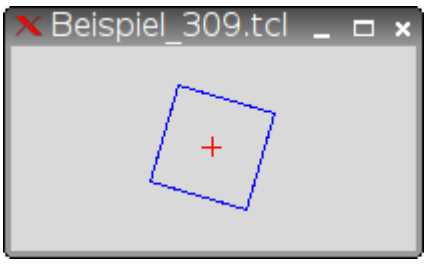
Die Zeilen 3 bis 22 enthalten die Prozedur, um die Eckpunkte des Polygons gemäß dem Drehwinkel und Drehpunkt umzurechnen. In den Zeilen 27 und 28 wird der Drehpunkt als kleines, rotes Kreuz markiert. In der Zeile 30 werden die Eckpunkte des Polygons festgelegt,

in diesem Fall ein Quadrat. In Zeile 32 werden die gedrehten Eckpunkte ermittelt. In Zeile 33 wird das gedrehte Quadrat gezeichnet.

Das folgende Beispiel dreht regelmäßig ein Quadrat, so dass eine Animation entsteht.

Listing 51.13: Einfache Animation (Beispiel309.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Drehen {Liste Grad DPx DPy} {
4    # Liste = Liste der Eckpunkt des Polygons
5    # Grad = Drehwinkel (0 bis 360 Grad), entgegen dem ↩
       Uhrzeigersinn
6    # DPx = x-Koordinate des Drehpunkts
7    # DPy = y-Koordinate des Drehpunkts
8
9    # Rueckgabe: Liste der transformierten Eckpunkte
10
11   set Pi 3.1415926535897931
12   set Grad2 [expr $Grad/180.0*$Pi]
13
14   set ListeGedreht {}
15   foreach {x y} $Liste {
16     set x2 [expr $x-$DPx]
17     set y2 [expr $y-$DPy]
18     lappend ListeGedreht [expr round($x2 * cos($Grad2) + ↩
         $y2 * sin($Grad2))+$DPx]
19     lappend ListeGedreht [expr round(-$x2 * sin($Grad2) + ↩
         $y2 * cos($Grad2))+$DPy]
20   }
21   return $ListeGedreht
22 }
23
24 canvas .c -width 200 -height 100
25 pack .c
26
27 set Linie1 [.c create line 95 50 105 50 -width 1 -fill red↩
     ] ; # Drehpunkt
28 set Linie2 [.c create line 100 45 100 55 -width 1 -fill ↩
     red] ; # Drehpunkt
29 set ListePunkte {}
30 set ListePunkte [list 75 25 125 25 125 75 75 75]
31
32 set Polygon [.c create polygon $ListePunkte -width 1 ↩
     -outline blue -fill ""]
33 for {set i 0} {$i <=360} {incr i} {
34   set Grad $i
35   set ListePunkteGedreht [Drehen $ListePunkte $Grad 100 ↩
       50]
36   .c coords $Polygon $ListePunkteGedreht
37   update idletasks
38   after 10
39 }
```



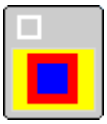
In Zeile 32 wird das Rechteck erzeugt. In den Zeilen 33 bis 39 wird das Quadrat gedreht. In Zeile 35 werden die neuen Koordinaten berechnet. In Zeile 36 wird das Quadrat gedreht. In Zeile 37 wird die Bildschirmausgabe aktualisiert. In Zeile 38 wird 10 Millisekunden gewartet. Bei dieser Form der Animation wartet das Programm solange, bis die Animation beendet ist. Das bedeutet, es werden während der Animation keine weiteren Befehle ausgeführt, insbesondere keine Interaktionen mit dem Anwender. Eine Animation, die während der Animation auf Anwenderaktionen reagiert, wird in Kapitel 53 auf Seite 629 beschrieben.

51.7 Farbe eines Bildpunkts ermitteln

Man kann auch die Farbe eines Bildpunkts ermitteln. Dazu muss man das das Paket `Img` verwenden, das standardmäßig in Tcl/Tk zur Verfügung steht.

Listing 51.14: Farbe eines Bildpunkts ermitteln (Beispiel481.tcl)

```
#!/usr/bin/env wish

package require Img

canvas .c -width 40 -height 30 -background yellow
pack .c
.c create rectangle 10 5 30 25 -width 5 -outline red -fill blue
update idletasks
tkwait visibility .c

set Bild [image create photo -format window -data .c]

set y 15
for {set x 1} {$x <= 40} {incr x} {
  set Pixel [$Bild get $x $y]
  puts "P($x/$y): $Pixel"
}

image delete $Bild
```



```
oli@debian: ~
oli@debian:~$ ./Beispiel481.tcl
P(1/15): 255 255 0
P(2/15): 255 255 0
P(3/15): 255 255 0
P(4/15): 255 255 0
P(5/15): 255 255 0
P(6/15): 255 255 0
P(7/15): 255 255 0
P(8/15): 255 0 0
P(9/15): 255 0 0
P(10/15): 255 0 0
P(11/15): 255 0 0
P(12/15): 255 0 0
P(13/15): 0 0 255
P(14/15): 0 0 255
P(15/15): 0 0 255
P(16/15): 0 0 255
P(17/15): 0 0 255
P(18/15): 0 0 255
P(19/15): 0 0 255
P(20/15): 0 0 255
P(21/15): 0 0 255
P(22/15): 0 0 255
P(23/15): 0 0 255
P(24/15): 0 0 255
P(25/15): 0 0 255
P(26/15): 0 0 255
P(27/15): 0 0 255
P(28/15): 255 0 0
P(29/15): 255 0 0
P(30/15): 255 0 0
P(31/15): 255 0 0
P(32/15): 255 0 0
P(33/15): 255 255 0
P(34/15): 255 255 0
P(35/15): 255 255 0
P(36/15): 255 255 0
P(37/15): 255 255 0
P(38/15): 255 255 0
P(39/15): 255 255 0
P(40/15): 255 255 0
oli@debian:~$
```

In Zeile 3 wird das Paket `Img` eingebunden. In den Zeilen 5 bis 7 wird eine Zeichnung erstellt. In Zeile 8 wird die Bildschirmausgabe aktualisiert und in Zeile 8 wartet das Programm, bis die Zeichenfläche auf dem Bildschirm sichtbar ist. Beide Zeilen sind sehr wichtig, weil sonst die Farbwerte schon ermittelt werden, bevor die Zeichnung sichtbar geworden ist. Das würde zu falschen Ergebnissen führen. In Zeile 11 wird ein Screenshot von der Zeichnung erstellt und in die Variable `Bild` gespeichert. In den Zeilen 13 bis 17 werden die Pixel aus der 15. Zeile der Zeichnung ausgelesen und die Farbwerte angezeigt. Die Werte sind RGB-Werte (Rot, Grün, Blau). In Zeile 19 wird die Variable `Bild` wieder geleert, um den Speicher freizugeben.

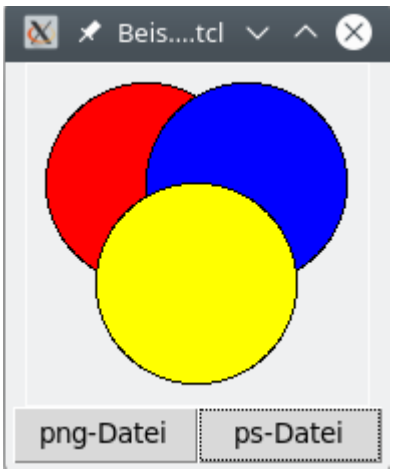
Die Koordinaten der Pixel beziehen sich auf den sichtbaren Teil der Zeichenfläche. Wenn ein Teil der Zeichenfläche nicht sichtbar ist, muss dieser Teil erst in den sichtbaren Bereich verschoben werden.

51.8 Bildschirmfoto erstellen

Man kann sehr leicht ein Bildschirmfoto erstellen und als Datei im png-Format oder Postscript-Format speichern.

Listing 51.15: Bildschirmfoto erstellen (Beispiel536.tcl)

```
#!/usr/bin/env wish

package require Img

proc ScreenshotPNG {Canvasname Dateiname} {
  set Screengrab [image create photo -format window -data $Canvasname]
  $Screengrab write $Dateiname -format png
  image delete $Screengrab
}

proc ScreenshotPS {Canvasname Dateiname} {
  $Canvasname postscript -file $Dateiname
}

canvas .c -width 170 -height 170
set KreisRot [.c create oval 10 10 110 110 -width 1 -outline black -fill red]
set KreisBlau [.c create oval 60 10 160 110 -width 1 -outline black -fill blue]
set KreisGelb [.c create oval 35 60 135 160 -width 1 -outline black -fill yellow]

ttk::frame .fr
ttk::button .fr.bt1 -text "png-Datei" -command {ScreenshotPNG .c Screenshot.png}
ttk::button .fr.bt2 -text "ps-Datei" -command {ScreenshotPS .c Screenshot.ps}
grid .fr.bt1 -row 0 -column 0
grid .fr.bt2 -row 0 -column 1

pack .c
pack .fr
```



Zuerst muss man das Packet `IMG` laden (Zeile 3). In den Zeilen 5 bis 9 sieht man die Prozedur zum Erstellen eines Bildschirmfotos im png-Format und in Zeile 12 den Befehl zum Erstellen des Bildschirmfotos im Postscript-Format.

51.9 Zeichenfläche mit Scroll-Leisten

Listing 51.16: Zeichenfläche mit Scroll-Leisten (Beispiel314.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::frame .fr
4  canvas .fr.c -width 200 -height 200 -xscrollcommand "
       .fr.xscroll set" -yscrollcommand ".fr.yscroll set"
5
6  set FotoID [image create photo Foto -file "foto2.ppm"]
7  .fr.c create image 0 0 -anchor nw -image Foto
8  .fr.c configure -scrollregion [.fr.c bbox all]
9
10 ttk::scrollbar .fr.xscroll -orient horizontal -command "
       .fr.c xview"
11 ttk::scrollbar .fr.yscroll -command ".fr.c yview"
12
13 pack .fr.xscroll -side bottom -fill x
14 pack .fr.yscroll -side right -fill y
15 pack .fr.c -expand yes -fill both -side top
16 pack .fr -expand yes -fill both -side top
```

In Zeile 4 wird die Zeichenfläche mit den Scroll-Leisten verknüpft. In Zeile 8 wird die Größe der Scroll-Region festgelegt. Dies ist in der Regel die gesamte Zeichenfläche. Die Größe der Zeichenfläche bestimmt man am einfachsten mit dem Befehl `[.fr.c bbox all]`. Der Befehl analysiert die Größe und Position aller Objekte auf der Zeichenfläche und ermittelt daraus die Größe der Zeichenfläche. Wenn man Objekte verschiebt, hinzufügt oder löscht, sollte man anschließend mit diesem Befehl die Größe der Zeichenfläche neu ermitteln. In den Zeilen 10 und 11 werden die Scroll-Leisten erzeugt.

51.10 Zeichenfläche, die größer als das Fenster ist

Wenn man beispielsweise ein Bild anzeigt, das größer als das Fenster ist, gibt es zwei Koordinatensysteme: zum einen die Koordinaten bezogen auf das Fenster und zum anderen die Koordinaten bezogen auf das Bild. Das nachfolgende Beispiel zeigt den Umgang mit Fenster- und Bildkoordinaten.

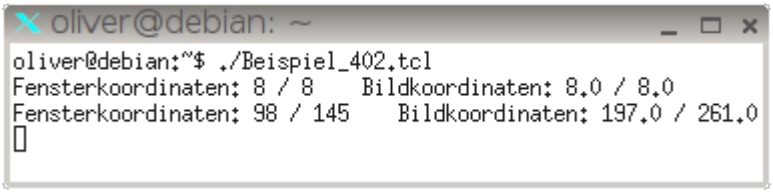
Listing 51.17: Fenster- und Bildkoordinaten (Beispiel402.tcl)

```
1  #!/usr/bin/env wish
2
3  set Konf(Skriptname) [info script]
4  if {[file type $Konf(Skriptname)] == "link"} {
5          set Konf(Skriptname) [file readlink $Konf(
               Skriptname)]
6  }
7
8  set Konf(Bild) [file join [file dirname $Konf(Skriptname)]
       foto2.ppm]
9
10 proc SichtbarerBildteil {BildBreite BildHoehe} {
11         # ermittelt die sichtbare Breite und Hoehe des
               Bildes (in Pixel)
12         # Rueckgabe: eine Liste bestehend aus der
               sichtbaren Breite und der sichtbaren Hoehe
13         set tmp [.fr.c xview]
14         set tmp1 [lindex $tmp 0]
```

```
15          set tmp2 [lindex $tmp 1]
16          set Breite [expr round(1.0*($tmp2 - $tmp1) * ↩
            $BildBreite)]
17
18          set tmp [.fr.c yview]
19          set tmp1 [lindex $tmp 0]
20          set tmp2 [lindex $tmp 1]
21          set Hoehe [expr round(1.0*($tmp2 - $tmp1) * ↩
            $BildHoehe)]
22
23          set Rueckgabe ""
24          lappend Rueckgabe $Breite
25          lappend Rueckgabe $Hoehe
26          return $Rueckgabe
27 }
28
29 proc BildZentrieren {x y BildBreite BildHoehe} {
30          # Verschiebt das Bild, so dass die Bild-Koordinate↩
             x/y in der Mitte des sichtbaren Fensters ist
31
32          set tmp [SichtbarerBildteil $BildBreite $BildHoehe↩
            ]
33          set FensterBreite [lindex $tmp 0]
34          set FensterHoehe [lindex $tmp 1]
35
36          set xMittelpunkt [expr max(1.0*($x - ↩
            $FensterBreite/2)/$BildBreite,0)]
37          set yMittelpunkt [expr max(1.0*($y - $FensterHoehe↩
            /2)/$BildHoehe,0)]
38          .fr.c xview moveto $xMittelpunkt
39          .fr.c yview moveto $yMittelpunkt
40 }
41
42 proc PunktZeichnen {x y Radius Farbe Dicke} {
43          # x/y = Bild-Koordinaten
44          set x1 [expr max($x-$Radius,0)]
45          set x2 [expr $x+$Radius]
46          set y1 [expr max($y-$Radius,0)]
47          set y2 [expr $y+$Radius]
48          set ID [.fr.c create oval $x1 $y1 $x2 $y2 -width ↩
            $Dicke -outline $Farbe -fill $Farbe]
49          return $ID
50 }
51
52 proc FindeItem {x y} {
53    set x [.fr.c canvasx $x]
54    set y [.fr.c canvasy $y]
55    set ID [.fr.c find closest $x $y]
56    return $ID
57 }
58
59 ttk::frame .fr
60 canvas .fr.c -width 100 -height 150 -xscrollcommand "↩
```

```
    .fr.xscroll set" -yscrollcommand ".fr.yscroll set"

set Bild2 [image create photo Bild -file $Konf(Bild)]
set BildBreite [image width $Bild2]
set BildHoehe [image height $Bild2]

.fr.c create image 0 0 -anchor nw -image Bild
.fr.c configure -scrollregion [.fr.c bbox all]
ttk::scrollbar .fr.xscroll -orient horizontal -command ".fr.c xview"
ttk::scrollbar .fr.yscroll -command ".fr.c yview"
pack .fr.xscroll -side bottom -fill x
pack .fr.yscroll -side right -fill y
pack .fr.c -expand yes -fill both -side top
pack .fr -expand yes -fill both -side top
bind .fr.c <ButtonRelease-1> {puts "Fensterkoordinaten: %x / %y    Bildkoordinaten: [%W canvasx %x] / [%W canvasy %y]"}
bind .fr.c <Shift-ButtonRelease-1> {set ID [FindeItem %x %y];.fr.c moveto $ID 40 60}
bind .fr.c <Control-ButtonRelease-1> {set ID [FindeItem %x %y];.fr.c moveto $ID 20 40}
bind .fr.c <ButtonRelease-3> {BildZentrieren 100 133 $BildBreite $BildHoehe}

set ID [PunktZeichnen 20 40 10 red 4]
# Das Bild ist 200 Pixel breit und 266 Pixel hoch
set MittelpunktID [PunktZeichnen 100 133 5 yellow 2]
```

Nach einem Klick auf die Ecke rechts unten:



Nach einem Klick mit der rechten Maustaste:

Das Bild hat eine Breite von 200 Pixel und eine Höhe von 266 Pixel. Das Fenster ist 100 Pixel breit und 150 Pixel hoch. Wenn man mit der linken Maustaste in die linke obere Ecke des Bildes klickt, bekommt man die Koordinaten angezeigt. In diesem Fall sind die Bildkoordinaten identisch mit den Fensterkoordinaten. Klickt man aber in die Ecke rechts unten, erhält man unterschiedliche Koordinaten. Der Code, der die beiden Koordinaten anzeigt steht in Zeile 74: `%x` bzw. `%y` enthält die Fensterkoordinaten. `[%W canvasx %x]` bzw. `[%W canvasy %y]` ermittelt die Bildkoordinaten
Wenn man die `Shift`-Taste gedrückt hält und dann mit der linken Maustaste auf den roten Punkt klickt, wird der Punkt im Bild versetzt. Mit der `Steuerung`-Taste plus linke Maustaste springt der Punkt wieder zurück. Der Code steht in den Zeilen 75 und 76 sowie in der Prozedur `FindeItem` (Zeilen 52 bis 57).
Wenn man mit der rechten Maustaste in das Bild klickt, wird das Bild zentriert. Der gelbe Punkt markiert die Bildmitte. Die Scroll-Leisten passen sich automatisch an. Der Code steht in der Zeile 77 sowie in den Prozeduren `BildZentrieren` (Zeilen 29 bis 40) und `SichtbarerBildteil` (Zeilen 10 bis 27).

51.11 Zeichenfläche als Hilfsmittel, viele Elemente in einem Fenster darzustellen

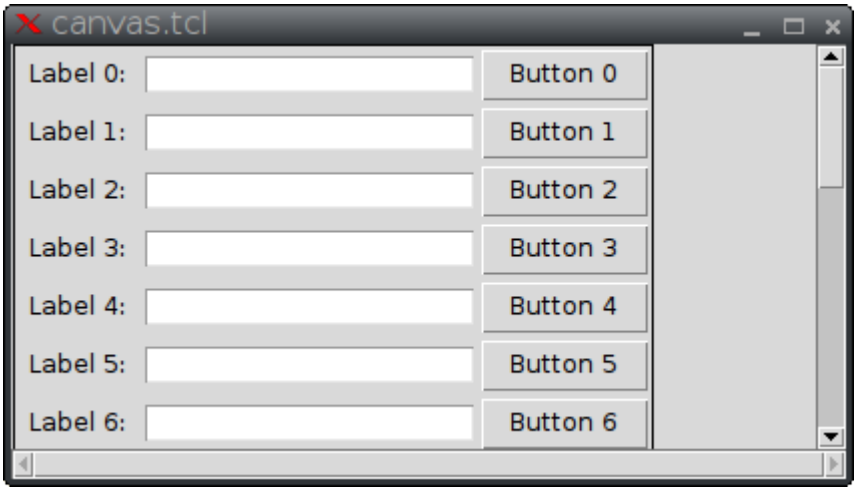
Es kann vorkommen, dass man in einem Fenster mehr Elemente anzeigen muss, als in die Fenstergröße (oder Bildschirmgröße) hineinpassen. Da das Fenster selbst keine Scroll-Leisten hat, kann der Anwender den Fensterinhalt nicht verschieben. In diesem Fall verwendet man ein Rahmen-Element, in das man ein Canvas-Element zusammen mit Scroll-Leisten platziert. In das Canvas-Element werden dann alle anderen Elemente gesetzt.

Listing 51.18: Dialog besteht komplett aus einer Zeichenfläche (Beispiel390.tcl)

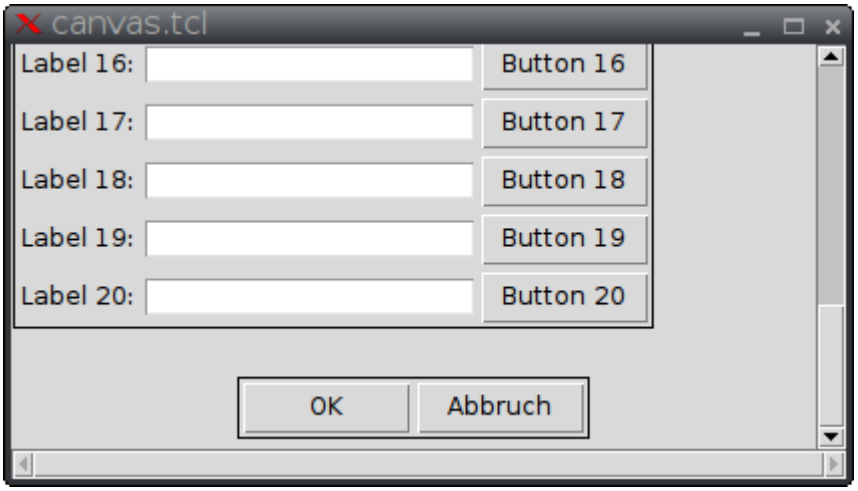
```
1 #!/usr/bin/env wish
2
3 # erzeugt einen Rahmen, der die Zeichenflaeche und die ↩
    Scroll-Leisten zusammenfasst
4 ttk::frame .frAlles
5
6 # erzeugt die Zeichenflaeche
7 canvas .frAlles.c -width 400 -height 200 -xscrollcommand "↩
    .frAlles.xscroll set" -yscrollcommand ".frAlles.yscroll↩
     set"
```

```
8  ttk::scrollbar .frAlles.xscroll -orient horizontal ↩
     -command ".frAlles.c xview"
9  ttk::scrollbar .frAlles.yscroll -command ".frAlles.c yview↩
     "
10 pack .frAlles.xscroll -side bottom -fill x
11 pack .frAlles.yscroll -side right -fill y
12 pack .frAlles.c -expand yes -fill both -side top
13
14 # erzeugt einen Rahmen mit allen Widgets
15 ttk::frame .frAlles.c.frWidgets -borderwidth 1 -relief ↩
     solid -width 340 -height 700
16 for {set i 0} {$i <=20} {incr i} {
17   ttk::label .frAlles.c.frWidgets.lb$i -text "Label $i:"
18   ttk::entry .frAlles.c.frWidgets.en$i
19   ttk::button .frAlles.c.frWidgets.bt$i -text "Button $i" ↩
       -command exit
20   grid .frAlles.c.frWidgets.lb$i -padx 2 -pady 2 -row $i ↩
       -column 0
21   grid .frAlles.c.frWidgets.en$i -padx 2 -pady 2 -row $i ↩
       -column 1
22   grid .frAlles.c.frWidgets.bt$i -padx 2 -pady 2 -row $i ↩
       -column 2
23 }
24
25 # erzeugt einen Rahmen mit allen Buttons, die fuer den ↩
     gesamten Dialog gelten
26 ttk::frame .frAlles.c.frButtons -borderwidth 1 -relief ↩
     solid -width 340 -height 40
27 ttk::button .frAlles.c.frButtons.btOK -text "OK" -command ↩
     exit
28 ttk::button .frAlles.c.frButtons.btAbbruch -text "Abbruch"↩
      -command exit
29 pack .frAlles.c.frButtons.btOK -padx 2 -pady 2 -side left
30 pack .frAlles.c.frButtons.btAbbruch -padx 2 -pady 2 -side ↩
     left
31
32 # platziert den Rahmen mit den Widgets und den Rahmen mit ↩
     den Buttons
33 .frAlles.c create window 0 0 -anchor nw -window ↩
     .frAlles.c.frWidgets
34 .frAlles.c create window 200 650 -anchor c -window ↩
     .frAlles.c.frButtons
35
36 # ermittelt die tatsaechliche Groesse der Zeichenflaeche
37 .frAlles.c configure -scrollregion [.frAlles.c bbox all]
38
39 # zeigt den Rahmen mit der Zeichenflaeche an
40 pack .frAlles -expand yes -fill both -side top
```

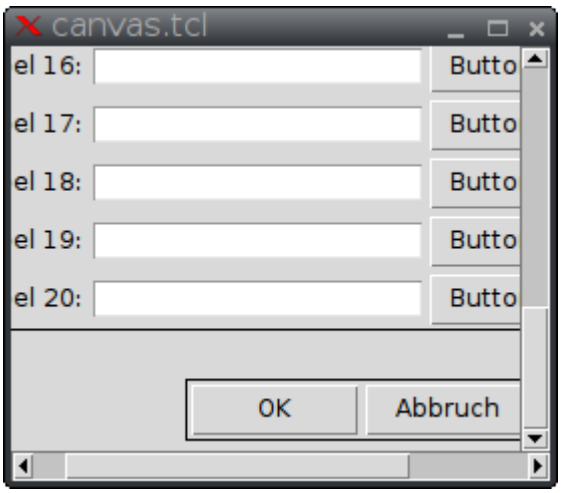
Das Fenster enthält mehr Elemente als bei gegebener Fenstergröße angezeigt werden können. Auch die Buttons zum gesamten Dialog sind zunächst nicht sichtbar:



Durch die vertikale Scroll-Leiste kann man bis zum letzten Element nach unten scrollen und erreicht auch die Buttons zum Beenden des Dialogs:



Wenn das Fenster verkleinert wird, bleiben durch die Scroll-Leisten alle Elemente weiterhin erreichbar:



In Zeile 4 wird ein Rahmen erzeugt, der sowohl die Zeichenfläche als auch die Scroll-Leisten aufnimmt. In den Zeilen 7 bis 12 wird die Zeichenfläche definiert. In Zeile 15 wird ein weiterer Rahmen erzeugt, der die zwanzig Eingabezeilen enthalten soll. Wichtig ist die Angabe der Optionen `-width` und `-height`. Daran wird erkannt, wann die Scroll-Leisten aktiviert werden müssen. In den Zeilen 16 bis 23 werden zwanzig Eingabezeilen, bestehend aus Label, Entry und Button erstellt. In Zeile 26 wird noch ein Rahmen erzeugt, der die Dialog-Buttons `OK` und `Abbruch` aufnimmt. Diese werden in den Zeilen 27 bis 30 definiert. In der Zeile 33 wird der Rahmen mit den Eingabezeilen auf der Zeichenfläche platziert. Der Befehl ist wie folgt aufgebaut:

```
Elementname create window x-Koordinate y-Koordinate
-anchor Ausrichtung -window Elementname
```

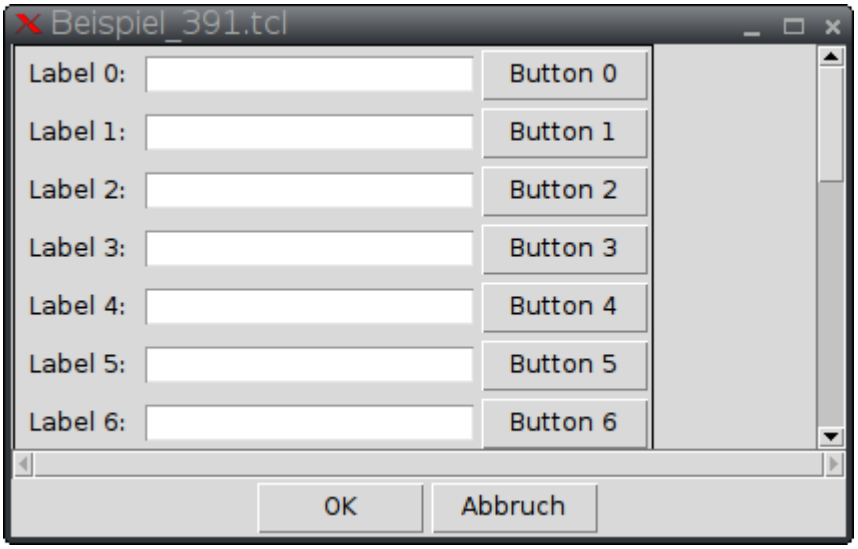
Der erste Elementname ist der Name der Zeichenfläche. Die Koordinaten für das einzufügende Fenster sind x=0 und y=0. Die Ausrichtung ist dieses Fensters ist links oben (`nw`=northwest). In dieses Fenster wird der Rahmen mit den Eingabezeilen eingefügt. In Zeile 34 wird der Rahmen mit den Dialog-Buttons eingefügt. In Zeile 37 wird jetzt die tatsächliche Größe der Zeichenfläche ermittelt und als Scroll-Region festgelegt. In Zeile 40 wird zum Schluss die Zeichenfläche mit ihren Scroll-Leisten und allen weiteren Elementen angezeigt.

Listing 51.19: Dialog besteht aus einer Zeichenfläche kombiniert mit Buttons, die immer sichtbar bleiben (Beispiel391.tcl)

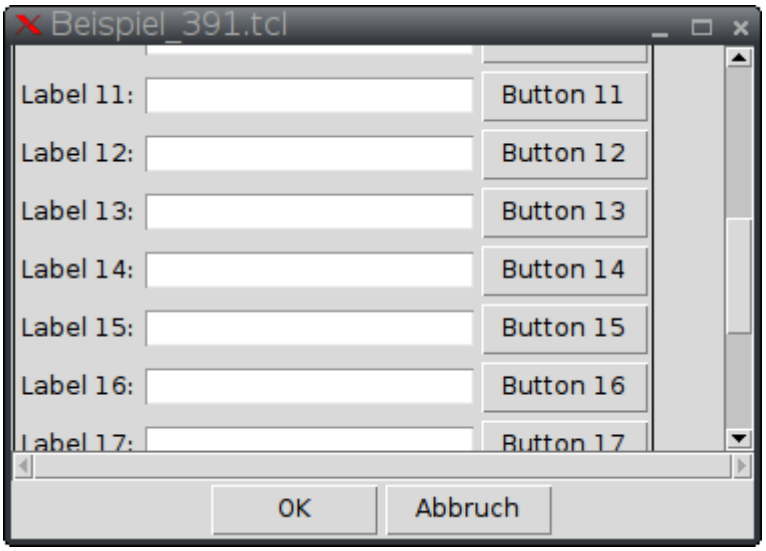
```
1 #!/usr/bin/env wish
2
3 # erzeugt einen Rahmen, der die Zeichenflaeche und die ↩
    Scroll-Leisten zusammenfasst
4 ttk::frame .frAlles
5
6 # erzeugt die Zeichenflaeche
7 canvas .frAlles.c -width 400 -height 200 -xscrollcommand "↩
    .frAlles.xscroll set" -yscrollcommand ".frAlles.yscroll↩
     set"
8 ttk::scrollbar .frAlles.xscroll -orient horizontal ↩
    -command ".frAlles.c xview"
9 ttk::scrollbar .frAlles.yscroll -command ".frAlles.c yview↩
```

```
    "
10 pack .frAlles.xscroll -side bottom -fill x
11 pack .frAlles.yscroll -side right -fill y
12 pack .frAlles.c -expand yes -fill both -side top
13
14 # erzeugt einen Rahmen mit allen Widgets
15 ttk::frame .frAlles.c.frWidgets -borderwidth 1 -relief ↩
    solid -width 340 -height 700
16 for {set i 0} {$i <=20} {incr i} {
17   ttk::label .frAlles.c.frWidgets.lb$i -text "Label $i:"
18   ttk::entry .frAlles.c.frWidgets.en$i
19   ttk::button .frAlles.c.frWidgets.bt$i -text "Button $i" ↩
       -command exit
20   grid .frAlles.c.frWidgets.lb$i -padx 2 -pady 2 -row $i ↩
       -column 0
21   grid .frAlles.c.frWidgets.en$i -padx 2 -pady 2 -row $i ↩
       -column 1
22   grid .frAlles.c.frWidgets.bt$i -padx 2 -pady 2 -row $i ↩
       -column 2
23 }
24
25 # platziert den Rahmen mit den Widgets
26 .frAlles.c create window 0 0 -anchor nw -window ↩
    .frAlles.c.frWidgets
27
28 # ermittelt die tatsaechliche Groesse der Zeichenflaeche
29 .frAlles.c configure -scrollregion [.frAlles.c bbox all]
30
31 # erzeugt einen Rahmen mit allen Buttons, die fuer den ↩
    gesamten Dialog gelten
32 ttk::frame .frButtons
33 ttk::button .frButtons.btOK -text "OK" -command exit
34 ttk::button .frButtons.btAbbruch -text "Abbruch" -command ↩
    exit
35 pack .frButtons.btOK -padx 2 -pady 2 -side left
36 pack .frButtons.btAbbruch -padx 2 -pady 2 -side left
37
38 # zeigt den Rahmen mit der Zeichenflaeche an und die ↩
    Buttons zum Dialog
39 pack .frAlles -expand yes -fill both -side top
40 pack .frButtons -side bottom
```

Das Fenster enthält mehr Elemente als bei gegebener Fenstergröße angezeigt werden können. Die Buttons zum gesamten Dialog sind aber trotzdem sichtbar:



Wenn man nach unten scrollt:



Das Beispiel ist weitgehend identisch mit dem vorherigen Beispiel. Lediglich die beiden Buttons `OK` und `Abbruch` sind nicht mehr Teil der Zeichenfläche, sondern werden als Unter-Elemente des Hautpfensters erstellt (Zeilen 31 bis 36). In den Zeilen 39 und 40 werden sowohl die Zeichenfläche als auch der Rahmen mit den Buttons angezeigt.

52 Gnuplot

Wenn man Gnuplot installiert hat, kann man sehr einfach Diagramme und Funktionen in Tcl/Tk darstellen.
Um ein Diagramm von Messwerten auszugeben, erstellen Sie zunächst eine Datei mit den Messwerten:

Beispiel311_data.txt

```
1;11;14
2;12;13
3;12;10
4;9;12
5;8;11
6;10;9
7;9;7
8;11;9
9;12;8
10;14;7
```

Danach erstellen Sie eine Datei mit dem Programmcode für Gnuplot:

Beispiel311_code.gnuplot

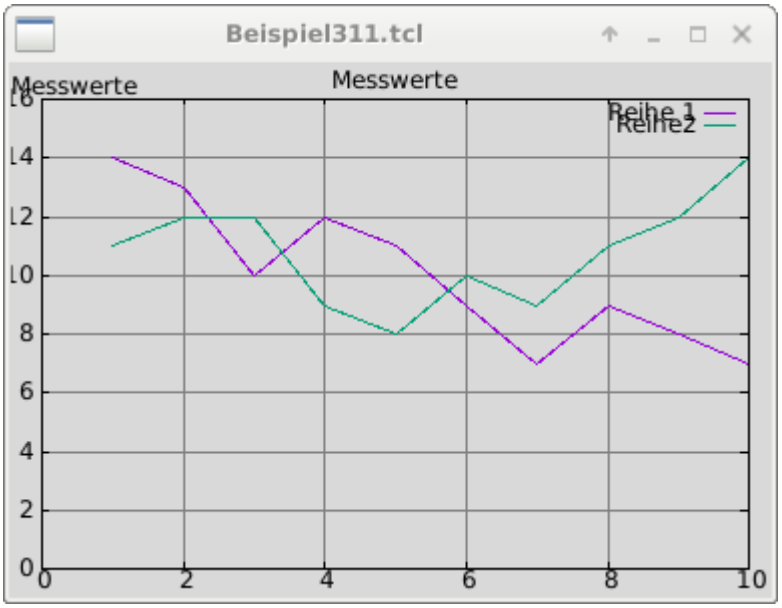
```
set terminal tkcanvas
set output 'canvas.tk'
set datafile separator ';'
set ytics 1
set xtics autofreq
set grid ytics xtics
set autoscale y
set autoscale x
set xrange [0:10]
set yrange [0:16]
set ytics 0,2,16
set ylabel 'Messwerte'
set title "Messwerte"
plot \
'Beispiel311_data.txt' using 1:3 title 'Reihe 1' with lines, \
'Beispiel311_data.txt' using 1:2 title 'Reihe2' with lines
```

Wichtig sind die beiden ersten Zeilen. Sie sorgen dafür, dass Gnuplot die Datei canvas.tk erstellt, die alle notwendigen Tcl/Tk-Befehle enthält, um später in Tcl/Tk das Diagramm erstellen zu können.
Nun schreiben Sie folgendes Tcl/Tk-Programm:

Listing 52.1: Messwerte aus einer Datei darstellen (Beispiel311.tcl)

```
#!/usr/bin/env wish

```

```
3 canvas .c
4 pack .c
5
6 exec gnuplot Beispiel311_code.gnuplot
7 source canvas.tk
8 file delete -force canvas.tk
9 gnuplot .c
```



In Zeile 6 wird das Programm Gnuplot ausgeführt. Gnuplot erzeugt die Datei `canvas.tk`, in der alle Tcl/Tk-Befehle zum Zeichnen des Diagramms stehen. In Zeile 7 wird diese Datei eingelesen und ausgeführt. In Zeile 8 wird die Datei gelöscht. In Zeile 9 wird der Befehl `gnuplot` ausgeführt. Dieser Befehl ist eine Prozedur, die zuvor von Gnuplot in die Datei `canvas.tk` geschrieben wurde und somit jetzt in Tcl/Tk zur Verfügung steht.

Wie man erkennt, sieht die Grafik nicht besonders ansprechend aus. Das Ergebnis ist deutlich besser, wenn Gnuplot nicht direkt das Canvas anspricht, sondern eine Grafikdatei im png-Format erzeugt. Hierzu ändert man die ersten beiden Zeilen des Gnuplot-Programmcodes:

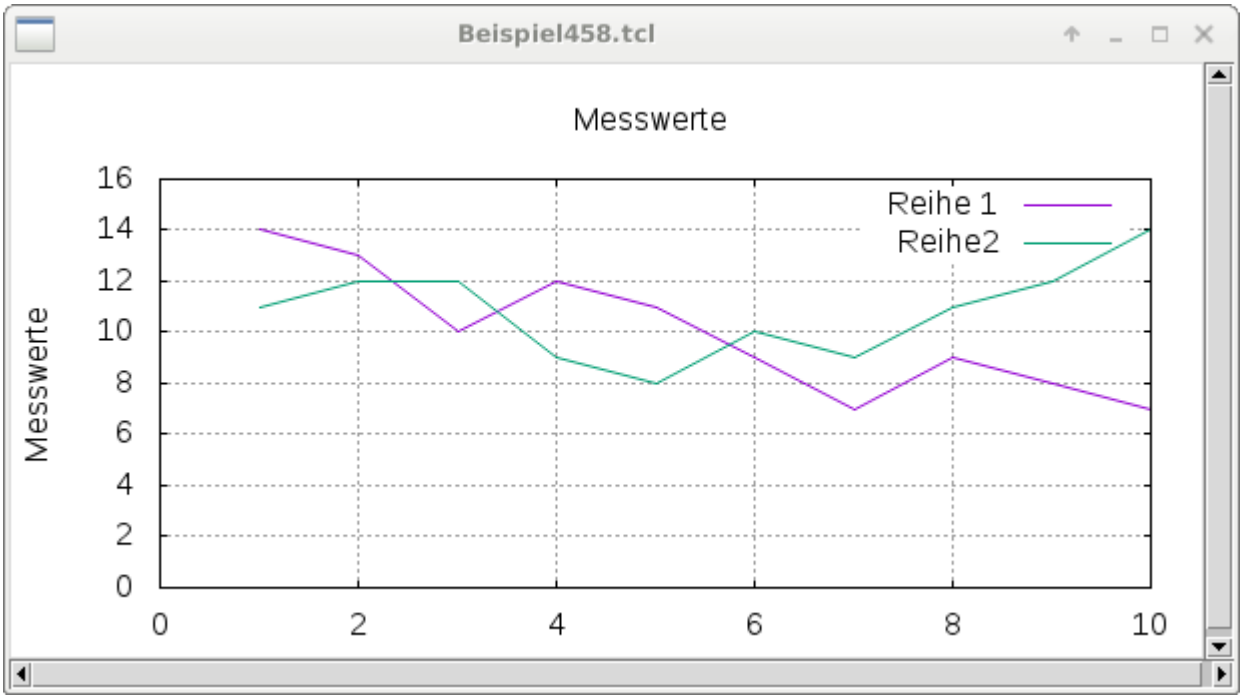
Beispiel458_code.gnuplot

```
set terminal png enhanced truecolor size 600, 300
set output 'Grafik.png'
set datafile separator ';'
set ytics 1
set xtics autofreq
set grid ytics xtics
set autoscale y
set autoscale x
set xrange [0:10]
set yrange [0:16]
set ytics 0,2,16
set ylabel 'Messwerte'
set title "Messwerte"

plot \
'Beispiel311_data.txt' using 1:3 title 'Reihe 1' with lines, \
'Beispiel311_data.txt' using 1:2 title 'Reihe2' with lines
```

Auch das Tcl/Tk-Programm wird geändert, so dass es die Grafikdatei anzeigt:

Listing 52.2: Messwerte aus einer Datei darstellen (mit Grafikdatei) (Beispiel458.tcl)

```
#!/usr/bin/env wish

exec gnuplot Beispiel458_code.gnuplot

ttk::frame .fr
canvas .fr.c -width 300 -height 150 -xscrollcommand ".fr.xscroll set" -yscrollcommand ".fr.yscroll set"

image create photo Bild -file Grafik.png

.fr.c create image 0 0 -anchor nw -image Bild
.fr.c configure -scrollregion [.fr.c bbox all]
ttk::scrollbar .fr.xscroll -orient horizontal -command ".fr.c xview"
ttk::scrollbar .fr.yscroll -command ".fr.c yview"
pack .fr.xscroll -side bottom -fill x
pack .fr.yscroll -side right -fill y
pack .fr.c -expand yes -fill both -side top
pack .fr -expand yes -fill both -side top
```



Gnuplot kann auch Funktionen darstellen. Hierzu erstellen Sie folgende Datei mit Gnuplot-Befehlen:

Beispiel312_code.gnuplot

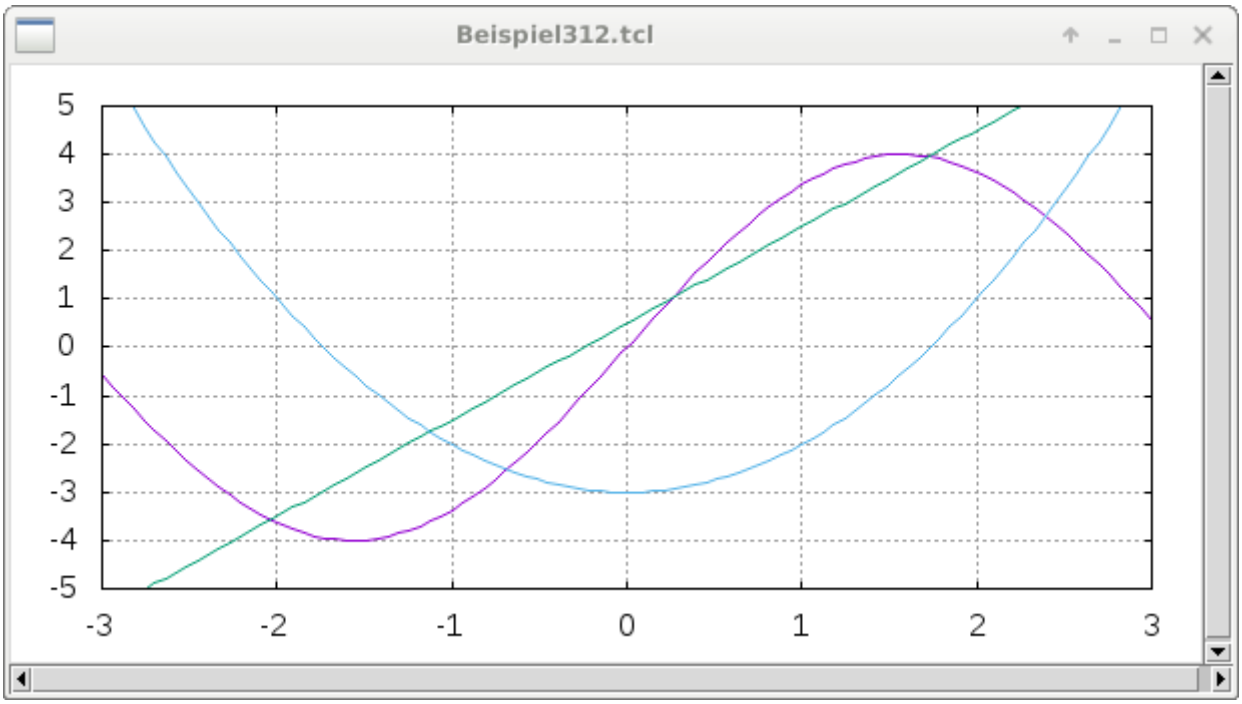
```
1  set terminal png enhanced truecolor size 600, 300
2  set output 'Grafik.png'
3  set ytics 1
4  set xtics 1
5  set grid ytics xtics
6  set xrange[-3:3]
7  set yrange[-5:5]
8  set grid
9  plot 4*sin(x) notitle,2*x+0.5 notitle,x*x-3 notitle
```

In Zeile 9 werden drei Funktionen definiert, die durch ein Komma getrennt sind.
Das Tcl/Tk-Programm sieht wie folgt aus:

Listing 52.3: Funktionen darstellen (Beispiel312.tcl)

```
1 #!/usr/bin/env wish
2
3 exec gnuplot Beispiel312_code.gnuplot
4
5 ttk::frame .fr
6 canvas .fr.c -width 300 -height 150 -xscrollcommand "
    .fr.xscroll set" -yscrollcommand ".fr.yscroll set"
7
8 image create photo Bild -file Grafik.png
9
```

```
.fr.c create image 0 0 -anchor nw -image Bild
.fr.c configure -scrollregion [.fr.c bbox all]
ttk::scrollbar .fr.xscroll -orient horizontal -command ".fr.c xview"
ttk::scrollbar .fr.yscroll -command ".fr.c yview"
pack .fr.xscroll -side bottom -fill x
pack .fr.yscroll -side right -fill y
pack .fr.c -expand yes -fill both -side top
pack .fr -expand yes -fill both -side top
```

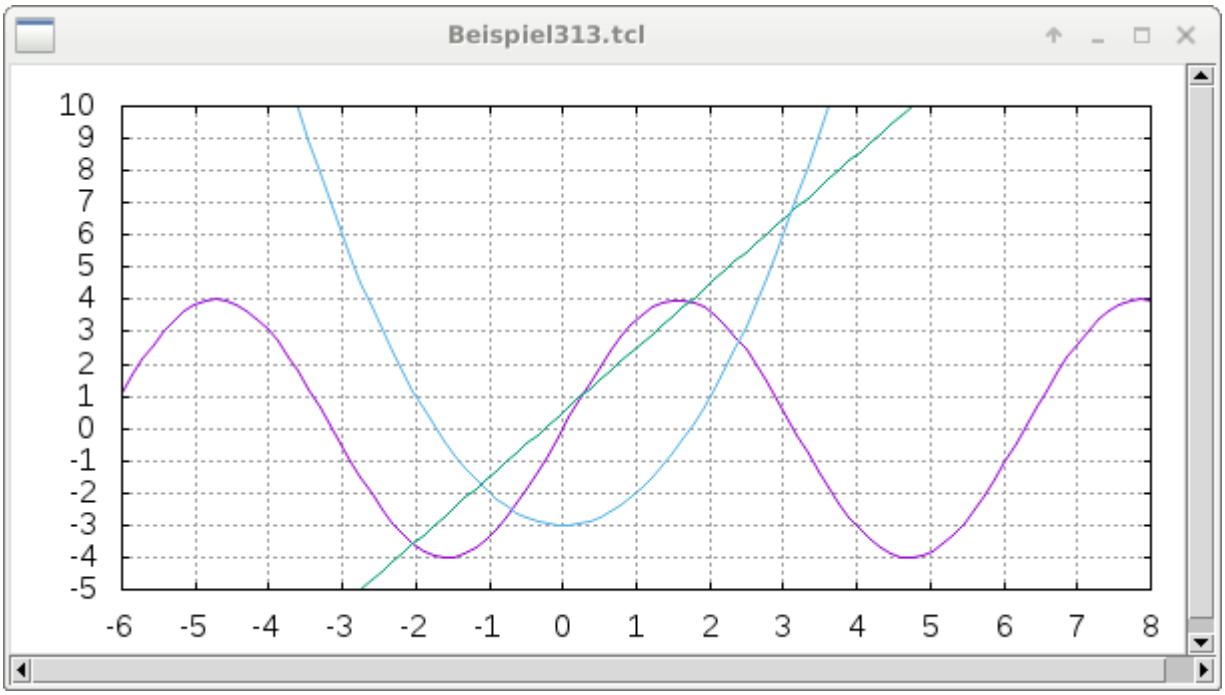


Man kann Gnuplot auch direkt aus Tcl/Tk heraus ansprechen. Damit erhält man die Möglichkeit, Variablenwerte an Gnuplot zu übergeben.

Listing 52.4: Variablenübergabe an Gnuplot (Beispiel313.tcl)

```
#!/usr/bin/env wish

set xMin -6
set xMax 8
set yMin -5
set yMax 10

set pid [open "| gnuplot" "w"]
puts $pid "set terminal png enhanced truecolor size 600, 300"
puts $pid "set output 'Grafik.png'"
puts $pid "set ytics 1"
puts $pid "set xtics 1"
puts $pid "set grid ytics xtics"
```

```
14 puts $pid "set xrange\[$xMin:$xMax\]"
15 puts $pid "set yrange\[$yMin:$yMax\]"
16 puts $pid "set grid"
17 puts $pid "plot 4*sin(x) notitle,2*x+0.5 notitle,x*x-3 notitle"
18 close $pid
19
20 ttk::frame .fr
21 canvas .fr.c -width 300 -height 150 -xscrollcommand ".fr.xscroll set" -yscrollcommand ".fr.yscroll set"
22
23 image create photo Bild -file Grafik.png
24
25 .fr.c create image 0 0 -anchor nw -image Bild
26 .fr.c configure -scrollregion [.fr.c bbox all]
27 ttk::scrollbar .fr.xscroll -orient horizontal -command ".fr.c xview"
28 ttk::scrollbar .fr.yscroll -command ".fr.c yview"
29 pack .fr.xscroll -side bottom -fill x
30 pack .fr.yscroll -side right -fill y
31 pack .fr.c -expand yes -fill both -side top
32 pack .fr -expand yes -fill both -side top
```



In den Zeilen 6 bis 9 werden Variablen definiert, die später die Achsenbereiche festlegen. In Zeile 11 wird eine sogenannte Pipe nach Gnuplot geöffnet. In den Zeilen 12 bis 20 werden die Gnuplot-Befehle übertragen. In den Zeilen 17 und 18 ist darauf zu achten, dass die eckigen Klammern `[` und `]` durch einen vorangestellten Schrägstrich `text` maskiert werden. Sonst kommt es zu einer Fehlermeldung. In Zeile 21 wird die Pipe geschlossen.

53 Animation erstellen

Befehle:

- `after`
- `after cancel`

Wenn man Animationen erstellt, muss man darauf achten, dass das Programm auch während der Animation auf Eingaben des Anwenders reagiert. Deshalb zerlegt man die Animation in kleine Einzelschritte, die nacheinander alle paar Millisekunden ausgeführt werden. Hierzu dient der Befehl `after`. Der Befehl ruft nach einer bestimmten Zeitdauer eine Prozedur auf, arbeitet die Prozedur ab und kehrt zum aufrufenden Programmteil zurück. Dabei handelt es sich nicht um eine parallele Verarbeitung der Befehle (Stichwort multithreading), sondern um eine sequentielle Verarbeitung. Man kann sich die Funktionsweise des Befehls als eine Art Aufgabenplanung vorstellen: es wird festgelegt, wann welche Aufgabe ausgeführt werden soll.

Listing 53.1: Animation eines Balls während der Eingabe (Beispiel315.tcl)

```
#!/usr/bin/env wish

proc Animation {dx dy} {
  # bewegt den Ball waehrend der Anwender eine Eingabe macht
  global Ball

  set Koordinaten [.c coords $Ball]
  set x [lindex $Koordinaten 0]
  set y [lindex $Koordinaten 1]

  # wenn der Ball an den Rand kommt, Richtung aendern
  if {$y >= 83} {
    set dy -1
  }
  if {$y <= 0} {
    set dy 1
  }
  if {$x >= 283} {
    set dx -1
  }
  if {$x <= 0} {
    set dx 1
  }

  .c move $Ball $dx $dy
  update idletasks
  after 10 Animation $dx $dy
```

```
28 }
29
30 proc AnimationStoppen {} {
31   # loescht alle bestehenden after-Aufrufe
32   foreach Element [after info] {
33     after cancel $Element
34   }
35 }
36
37 canvas .c -width 300 -height 100 -background yellow
38 pack .c
39 set Ball [.c create oval 0 0 16 16 -width 1 -outline black
     -fill black]
40
41 ttk::frame .fr
42 ttk::label .fr.lb -text "Eingabe:"
43 ttk::entry .fr.en
44 ttk::button .bt -text "Animation starten" -command {
    Animation 1 1}
45
46 pack .bt
47 pack .fr.lb -side left
48 pack .fr.en -side left
49 pack .fr
50
51 bind . "<Return>" AnimationStoppen
52 bind . "<KP_Enter>" AnimationStoppen
```

Starten Sie das Programm und klicken Sie auf den Button `Animation starten`. Der Ball bewegt sich im gelben Rechteck. Dennoch können Sie im Eingabefeld eine Eingabe machen. Wenn Sie die `Return`-Taste drücken, stoppt die Animation.
In den Zeilen 3 bis 28 wird die Animation des Balls durchgeführt. Die Prozedur sorgt in Zeile 27 dafür, dass die Prozedur nach 10 Millisekunden erneut aufgerufen wird. Es ist wichtig zu wissen, dass es sich um keinen rekursiven Aufruf handelt: der zweite Prozeduraufruf ist kein Unterprozess des ersten Prozeduraufrufs. In den Zeilen 30 bis 35 wird die Animation gestoppt. Dazu werden alle mit dem `after`-Befehl eingeplanten, aber noch nicht ausgeführten Befehle (also Aufrufe der Prozedur `Animation`) gelöscht. In den Zeilen 51 und 52 werden die Tastatur-Ereignisse `Return`-Taste und `Enter`-Taste mit der Prozedur `AnimationStoppen` verknüpft.

54 Layout

54.1 Grundlagen

Bei den klassischen Elementen wird das Layout (Schrift, Farbe, Abstände usw.) bei jedem Element einzeln angegeben. Dies erschwert ein einheitliches Layout im gesamten Programm und erhöht den Aufwand, wenn das Layout nachträglich geändert wird.

Die neuen ttk-Elemente werden stattdessen über Themen und Stile formatiert. Ein Stil legt das Aussehen einer Element-Klasse fest. So gibt es einen Stil für Buttons, einen Stil für Labels usw. Alle Stile zusammen bilden ein Thema. Es gibt verschiedene Themen (`default`, `classic`, `clam` usw.) zur Auswahl, deren Aussehen vom jeweiligen Betriebssystem abhängt. Man kann immer nur ein Thema gleichzeitig ausgewählt haben.

Auf Basis eines Themas und der zugehörigen Stile kann man ein eigenes Layout definieren. Dabei muss man nur die vom Basis-Thema bzw. Basis-Stil abweichenden Optionen festlegen. Wenn man Programme unter verschiedenen Betriebssystemen ausführen möchte, kann es allerdings vorteilhafter sein kein Thema festzulegen. Das sollte man einmal ausprobieren.

Tabelle 54.1: Die wichtigsten Befehle

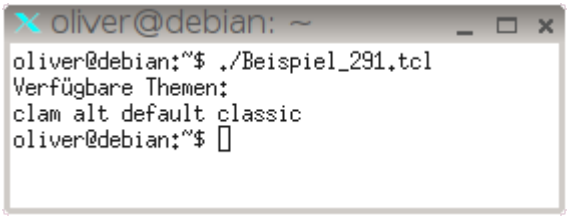
Befehl	Beschreibung
`puts [ttk::style theme names]`	Verfügbare Themen anzeigen
`ttk::style theme use Thema`	Ein Thema auswählen
`puts [Elementname cget -style]`	Stil anzeigen
`Elementname configure -style Stil`	Stil einem Element zuweisen
`puts [winfo class Elementname]`	Klasse anzeigen, zu der das Element gehört
`puts [ttk::style layout Klasse]`	Einzelelemente der Klasse anzeigen
`puts [ttk::style element options Einzelelement]`	Optionen eines Einzelelements anzeigen
`ttk::style configure Einzelelement Option Wert`	Eine Option des Einzelelements setzen
`puts [ttk::style lookup Einzelelement Option]`	Den Wert einer Option des Einzelelements anzeigen

Tabelle 54.1: Die wichtigsten Befehle

Befehl	Beschreibung
`ttk::style configure Stil.Klasse Option Wert Option Wert`	Einen eigenen Stil definieren
`ttk:Klasse Elementname -style Stil`	Einem Element den eigenen Stil zuordnen

Listing 54.1: Verfügbare Themen anzeigen und auswählen (Beispiel291.tcl)

```
1 #!/usr/bin/env wish
2
3 puts "Verfuegbare Themen:"
4 puts [ttk::style theme names]
5 ttk::style theme use classic
```



In Zeile 4 werden die verfügbaren Themen abgefragt. In Zeile 5 wird ein Thema ausgewählt.

Listing 54.2: Stil eines Elements abfragen und zuweisen (Beispiel292.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::button .b1 -text "Button 1"
4 ttk::button .b2 -text "Button 2" -style MeinStil.TButton
5
6 puts "Stil von Button 1:"
7 puts [.b1 cget -style]
8
9 puts "Stil von Button 2:"
10 puts [.b2 cget -style]
11
12 .b2 configure -style NeuerStil.TButton
13 puts "Neuer Stil von Button 2:"
14 puts [.b2 cget -style]
```

```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_292.tcl
Stil von Button 1:

Stil von Button 2:
MeinStil.TButton
Neuer Stil von Button 2:
NeuerStil.TButton
```

In den Zeilen 3 und 4 werden zwei Buttons definiert. Der Button `.bt2` bekommt direkt einen Stil zugeordnet. In Zeile 7 wird der Stil von Button `.bt1` abgefragt. Da kein Stil zugeordnet wurde, ist das Ergebnis leer. In Zeile 10 wird der Stil von Button `.bt2` abgefragt. In Zeile 12 wird dem Button `.bt2` ein neuer Stil zugeordnet. Man erkennt, dass man einem Element einen Stil zuordnen kann, obwohl der Stil noch nicht definiert wurde (der Stil `MeinStil.TButton` wurde im Programm an keiner Stelle definiert). Wenn das Programm den Stil nicht findet, wird immer auf einen default-Stil zurückgegriffen. Es erscheint keine Fehlermeldung. Leider gibt es keinen Befehl, der alle Stile zu allen Element-Klassen anzeigt.

Tabelle 54.2: Element und seine Klasse

Element	Klasse
`Button`	`TButton`
`Checkbutton`	`TCheckbutton`
`Combobox`	`TCombobox`
`Entry`	`TEntry`
`Frame`	`TFrame`
`LabelFrame`	`TLabelFrame`
`Menubutton`	`TMenubutton`
`Notebook`	`TNotebook`
`PanedWindow`	`TPanedwindow` (nicht `TPanedWindow` !)
`Progressbar`	`Horizontal.TProgressbar` `Vertical.TProgressbar`
`Radiobutton`	`TRadiobutton`
`Scale`	`Horizontal.TScale` `Vertical.TScale`
`Scrollbar`	`Horizontal.TScrollbar` `Vertical.TScrollbar`
`Separator`	`TSeparator`

Tabelle 54.2: Element und seine Klasse

Element	Klasse
`Sizegrip`	`TSizegrip`
`Treeview`	`Treeview` (nicht `TTreeview` !)

Listing 54.3: Layout einer Element-Klasse ändern (Beispiel293.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::button .b -text "Button"
4  pack .b
5
6  puts "Der Button gehoert zu folgender Klasse:"
7  puts [winfo class .b]
8
9  puts "Der Button besteht aus folgenden (Einzel-)Elementen:"
10 puts [ttk::style layout TButton]
11
12 puts "Das (Einzel-)Element Button.label hat folgende Optionen:"
13 puts [ttk::style element options Button.label]
14
15 ttk::style configure TButton.label -font "helvetica 24"
16
17 puts "Die Option -font hat folgenden Wert:"
18 puts [ttk::style lookup TButton.label -font]
```

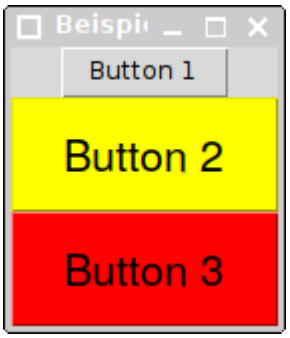
```
oliver@debian: ~
oliver@debian:~$ ./Beispiel_293.tcl
Der Button gehört zu folgender Klasse:
TButton
Der Button besteht aus folgenden (Einzel-)Elementen:
Button.border -sticky nswe -border 1 -children {Button.focus -sticky n
swe -children {Button.padding -sticky nswe -children {Button.label -st
icky nswe}}}
Das (Einzel-)Element Button.label hat folgende Optionen:
-compound -space -text -font -foreground -underline -width -anchor -ju
stify -wraplength -embossed -image -stipple -background
Die Option -font hat folgenden Wert:
helvetica 24
oliver@debian:~$
```

In Zeile 7 wird zunächst ermittelt, welcher Klasse das Element angehört. In Zeile 10 wird angezeigt, aus welchen einzelnen Elementen ein Button besteht und wie diese Elemente

hierarchisch angeordnet sind. Man erkennt, dass es ein Element `Button.border` gibt. Dieses Element hat das Unterelement `Button.focus`. Darunter gibt es das Element `Button.padding` und schließlich das Element `Button.label`. In Zeile 13 werden die Optionen des Elements `Button.label` abgefragt. Es gibt z. B. eine Option `-font`, mit der die Schriftart gesetzt werden kann. In Zeile 15 wird die Schriftart für die Klasse der Buttons definiert. Alle Buttons erhalten die Schriftart Helvetica und die Schriftgröße 24. In Zeile 18 wird der (eingestellte) Wert der Option `-font` angezeigt.
Es kann sein, dass man einen oder mehrere Buttons abweichend von den restlichen Buttons im Programm anders gestalten möchte.

Listing 54.4: Verschiedene Layouts (Beispiel294.tcl)

```
1  #!/usr/bin/env wish
2
3  ttk::style configure MeinStil.TButton -background yellow ↩
     -font "helvetica 16" -padding {10 10 10 10}
4  ttk::style configure Alarm.MeinStil.TButton -background ↩
     red
5
6  ttk::button .b1 -text "Button 1"
7  ttk::button .b2 -text "Button 2" -style MeinStil.TButton
8  ttk::button .b3 -text "Button 3" -style ↩
     Alarm.MeinStil.TButton
9  pack .b1
10 pack .b2
11 pack .b3
```



In Zeile 3 wird der Stil `MeinStil.TButton` definiert. Abweichend von dem Basis-Stil `TButton` soll der Stil eine gelbe Hintergrundfarbe, eine größere Schriftart und einen größeren Abstand zum Rand haben. In Zeile 4 wird auf Basis dieses Stils ein weiterer Stil `Alarm.MeinStil.TButton` definiert. Er erbt alle Eigenschaften vom Stil `MeinStil.TButton`, hat aber eine andere Hintergrundfarbe. In Zeile 7 wird dem Button `.b2` der Stil `MeinStil.TButton` zugeordnet, in Zeile 8 dem Button `.b3` der Stil `Alarm.MeinStil.TButton`.

Man kann das Layout sehr tiefgreifend beeinflussen. So hat ein Button zum Beispiel den Status `Maus ist über dem Button` oder `Button ist gedrückt`. Auch diesem Status kann ein Layout zugewiesen werden.

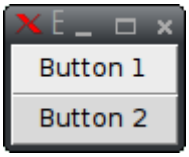
Tabelle 54.3: Status eines Elements

Status	Beschreibung
`active`	Die Maus ist in dem Element.
`background`	Nur unter Windows und MacOS. Das Element ist auf einem Fenster, das nicht im Vordergrund ist.
`disabled`	Das Element ist nicht aktiv.
`focus`	Das Element hat den Fokus.
`invalid`	Der Inhalt des Elements ist nicht gültig.
`pressed`	Das Element wird gerade gedrückt oder angeklickt.
`readonly`	Das Element kann nicht geändert werden.
`selected`	Das Element ist gerade ausgewählt.

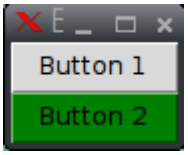
Listing 54.5: Layout eines einzelnen Elements in Abhängigkeit von seinem Status ändern (Beispiel295.tcl)

```
#!/usr/bin/env wish

ttk::style map MeinStil.TButton -background [list active green focus yellow] -foreground [list pressed red]

ttk::button .b1 -text "Button 1"
ttk::button .b2 -text "Button 2" -style MeinStil.TButton
pack .b1
pack .b2
```

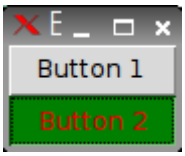
Nach dem Programmstart:



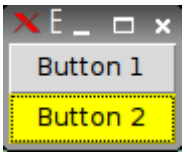
Wenn die Maus über den `Button 2` fährt:



Wenn der `Button 2` geklickt wird:



Nachdem der `Button 2` geklickt wurde und die Maus wieder auf `Button 1` steht:



Fahren Sie mit der Maus über den `Button 2` und klicken Sie ihn an. Ziehen Sie die Maus vom `Button 2` weg und wechseln Sie mit der Tabulator-Taste zwischen den Buttons.
In Zeile 3 werden folgende Einstellungen festgelegt: Die Hintergrundfarbe (`background`) ist grün, wenn die Maus über dem Button ist (`active`). Sie ist gelb, wenn der Button den Fokus hat. Die Vordergrundfarbe (`foreground`) wechselt auf rot, wenn der Button gedrückt wird (`pressed`).

54.2 Schrift

Um den Programmcode noch besser pflegen zu können, sollten auch die Schrifteigenschaften nicht bei jedem Stil einzeln festgelegt werden. Stattdessen sollte man Schriftart, -größe und sonstige Eigenschaften (fett, unterstrichen) unter einem Schriftnamen zusammenfassen und dann dem Element zuordnen.

Tabelle 54.4: Schriftoptionen

Option	Beschreibung

`-familiy Helvetica`
`-family Times`
`-family Courier`
	Schriftart
	Die drei genannten Schriftarten sind unabhängig vom Betriebssystem immer verfügbar und werden von Tcl/Tk automatisch durch eine passende Schrift des Computers ersetzt. Dabei steht Helvetica für eine serifenlose Schrift, Times für eine Schrift mit Serifen und Courier für eine proportionale Schrift.

`-weight normal` `-weight bold`	normal oder fett (bold)

Tabelle 54.4: Schriftoptionen

Option	Beschreibung
`-slant normal` `-slant italic`	normal oder kursiv (italic)

Listing 54.6: Schriftname verwenden (Beispiel296.tcl)

```
1 #!/usr/bin/env wish
2
3 font create MeineSchrift -family Helvetica -size 14 ↩
    -weight bold -slant italic
4 ttk::style configure MeinStil.TButton -font MeineSchrift ↩
    -padding {10 10 10 10}
5
6 ttk::button .b1 -text "Button 1"
7 ttk::button .b2 -text "Button 2" -style MeinStil.TButton
8 pack .b1
9 pack .b2
```

In Zeile 3 wird ein Schriftname festgelegt. In Zeile 4 wird dem Stil `MeinStil.TButton` die Schrift zugeordnet.

54.3 Farbe und Schrift der Elemente festlegen

Im Folgenden wird gezeigt, wie man bei den einzelnen ttk-Elementen die Farbe und Schriftgröße festlegt. Andere Eigenschaften wie das Mauszeigersymbol (`-cursor`), Abstände (`-padding`), Dicke von Rändern (`-borderwidth`) usw. sollen der Übersichtlichkeit wegen nicht betrachtet werden. Sie können ebenfalls für einen Stil festgelegt werden.

Wenn man sehen möchte, welche Farbe ein bestimmter Teil des Elements hat, ist es ratsam, in den Beispielen die Farben einzeln durch `red` auszutauschen.

Damit man schneller erkennt, an welchen Stellen in den folgenden Beispielen die Schrift eingestellt wird, werden nach dem Programmcode die relevanten Programmzeilen benannt.

54.3.1 Fenster

Um die Hintergrundfarbe des Programmfensters zu ändern, kann man ein `frame`-Element platzieren„ das automatisch seine Größe dem Fenster anpasst.

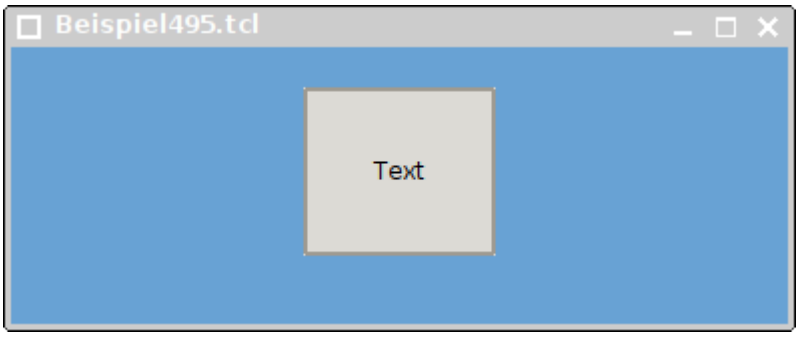
Listing 54.7: Ohne frame-Element (Beispiel494.tcl)

```
#!/usr/bin/env wish

ttk::style theme use clam

ttk::label .lb -text "Text" -borderwidth 5 -relief solid ↲
   -padding {30 30 30 30}
pack .lb -padx 20 -pady 20
```

Wie man sieht, hat das Fenster eine graue Hintergrundfarbe.

Listing 54.8: Mit frame-Element (Beispiel494.tcl)

```
1 #!/usr/bin/env wish
2
3 ttk::style theme use clam
4
5 ttk::label .lb -text "Text" -borderwidth 5 -relief solid ↲
     -padding {30 30 30 30}
6 pack .lb -padx 20 -pady 20
```



Durch das Platzieren eines `frame`-Elements (Zeilen 9-11) wird dem gesamten Fenster eine individuelle Hintergrundfarbe gegeben.

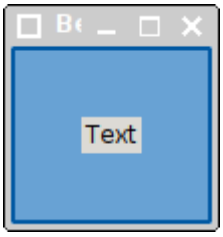
54.3.2 Frame

Listing 54.9: Frame (Beispiel482.tcl)

```
#!/usr/bin/env wish
```

```
2
3 set Farbe1 #0058A2 ; # dunkelblau
4 set Farbe2 #68A2D4 ; # mittelblau
5
6 ttk::style theme use clam
7
8 ttk::style configure MeinStil.TFrame -background $Farbe2
9 ttk::style configure MeinStil.TFrame -bordercolor $Farbe1 ; # nur verfuegbar, wenn das Thema clam (siehe Zeile 6) gewaehlt ist
10
11 ttk::frame .fr -padding {30 30 30 30} -borderwidth 5 -relief solid -style MeinStil.TFrame
12 ttk::label .fr.lb -text "Text"
13 pack .fr.lb -side top
14 pack .fr
```

Das `label`-Element in Zeile 12 wird nur benötigt, damit der Rahmen eine sichtbare Größe hat und soll in diesem Beispiel keine weitere Beachtung bekommen.

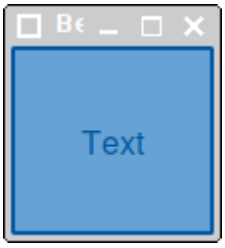


54.3.3 Label

Listing 54.10: Label (Beispiel483.tcl)

```
1 #!/usr/bin/env wish
2
3 set Farbe1 #0058A2 ; # dunkelblau
4 set Farbe2 #68A2D4 ; # mittelblau
5
6 font create MeineSchrift -family Helvetica -size 12 -slant roman -weight normal
7
8 ttk::style theme use clam
9
10 ttk::style configure MeinStil.TLabel -background $Farbe2
11 ttk::style configure MeinStil.TLabel -foreground $Farbe1
12 ttk::style configure MeinStil.TLabel -bordercolor $Farbe1 ; # nur verfuegbar, wenn das Thema clam (siehe Zeile 8) gewaehlt ist
13 ttk::style configure MeinStil.TLabel -font MeineSchrift
14
15 ttk::label .lb -text "Text" -style MeinStil.TLabel -borderwidth 5 -relief solid -padding {30 30 30 30}
16 pack .lb
```

Einstellung der Schrift: Zeilen 6 und 13.



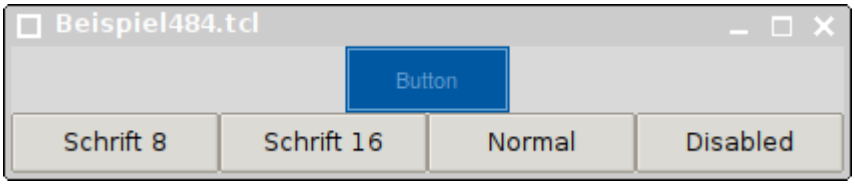
54.3.4 Button

Listing 54.11: Button (Beispiel484.tcl)

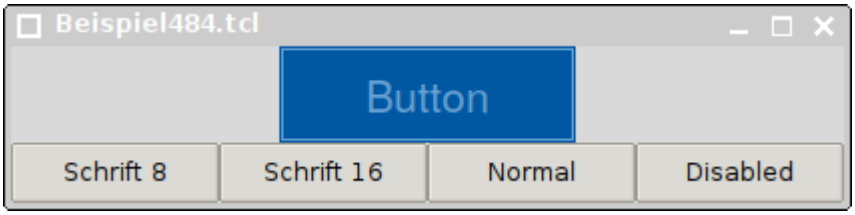
```
1  #!/usr/bin/env wish
2
3  set Farbe1 #0058A2 ; # dunkelblau
4  set Farbe2 #68A2D4 ; # mittelblau
5  set Farbe3 #DEE8F7 ; # hellblau
6  set Farbe4 #FFFFFF ; # weiss
7  set Farbe6 #D0D0D0 ; # hellgrau
8  set Farbe7 #A0A0A0 ; # mittelgrau
9  set Farbe8 #707070 ; # dunkelgrau
10 set Farbe10 #000000 ; # schwarz
11
12 font create MeineSchrift -family Helvetica -size 12 -slant
       roman -weight normal
13
14 ttk::style theme use clam
15
16 ttk::style configure MeinStil.TButton -background $Farbe1
17 ttk::style configure MeinStil.TButton -foreground $Farbe2
18 ttk::style configure MeinStil.TButton -bordercolor $Farbe1
       ; # Umrandung. Nur verfuegbar, wenn das Thema clam (
       siehe Zeile 14) gewaehlt ist.
19 ttk::style configure MeinStil.TButton -lightcolor $Farbe2 
       ; # Der Bereich zwischen der Umrandung und dem 
       Eingabebereich. Nur verfuegbar, wenn das Thema clam (
       siehe Zeile 14) gewaehlt ist.
20 ttk::style configure MeinStil.TButton -darkcolor $Farbe2 ;
       # Der Bereich zwischen der Umrandung und dem 
       Eingabebereich. Nur verfuegbar, wenn das Thema clam (
       siehe Zeile 14) gewaehlt ist.
21 ttk::style configure MeinStil.TButton -font MeineSchrift
22
23 ttk::style map MeinStil.TButton -background [list active 
       $Farbe2 disabled $Farbe6]
24 ttk::style map MeinStil.TButton -foreground [list active 
       $Farbe1 disabled $Farbe8]
25 ttk::style map MeinStil.TButton -bordercolor [list active 
       $Farbe1 disabled $Farbe7] ; # Nur verfuegbar, wenn das 
       Thema clam (siehe Zeile 14) gewaehlt ist.
```

```
26 ttk::style map MeinStil.TButton -lightcolor [list active $Farbe1 disabled $Farbe7] ; # Nur verfuegbar, wenn das Thema clam (siehe Zeile 14) gewaehlt ist.
27 ttk::style map MeinStil.TButton -darkcolor [list active $Farbe1 disabled $Farbe7] ; # Nur verfuegbar, wenn das Thema clam (siehe Zeile 14) gewaehlt ist.
28
29 ttk::button .bt -text "Button" -style MeinStil.TButton
30 pack .bt -side top
31
32 ttk::button .btKlein -text "Schrift 8" -command {font configure MeineSchrift -size 8}
33 ttk::button .btGross -text "Schrift 16" -command {font configure MeineSchrift -size 16}
34 pack .btKlein -side left
35 pack .btGross -side left
36
37 ttk::button .btNormal -text "Normal" -command {.bt configure -state normal}
38 ttk::button .btDisabled -text "Disabled" -command {.bt configure -state disabled}
39 pack .btNormal -side left
40 pack .btDisabled -side right
```

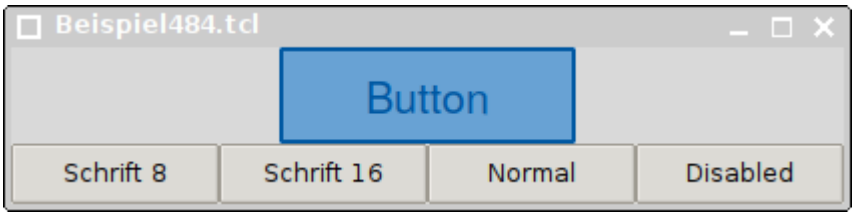
Einstellung der Schrift: Zeilen 12, 21, 32 und 33.
Klick auf den Button `Schrift 8`:



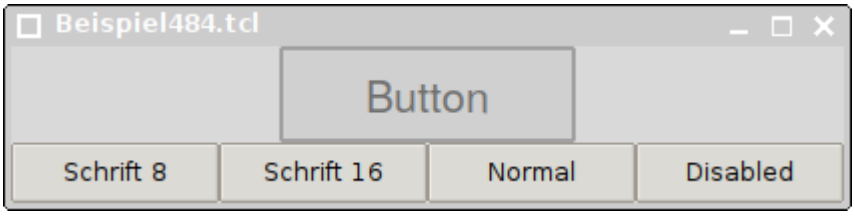
Klick auf den Button `Schrift 16`:



Maus ist über dem Button:



Klick auf den Button `Disabled`:



Wenn man die Schrift ändert, muss man sich überlegen, ob die Schrift für alle Elemente oder nur für eine bestimmte Klasse von Elementen (z. B. nur für Buttons) geändert werden soll.

Listing 54.12: Schrift wird für alle Elemente geändert (Beispiel485.tcl)

```
#!/usr/bin/env wish

font create MeineSchrift -family Helvetica -size 12 -slant roman -weight normal

ttk::style theme use clam

ttk::style configure MeinStil.TLabel -font MeineSchrift
ttk::style configure MeinStil.TButton -font MeineSchrift

ttk::label .lb -text "Label" -style MeinStil.TLabel
ttk::button .bt -text "Button" -style MeinStil.TButton
pack .lb -side top
pack .bt -side top

ttk::button .btKlein -text "Schrift 8" -command {font configure MeineSchrift -size 8}
ttk::button .btGross -text "Schrift 20" -command {font configure MeineSchrift -size 20}
pack .btKlein -side left
pack .btGross -side left
```

Klick auf den Button `Schrift 8`:

Klick auf den Button `Schrift 20`:

In den Zeilen 15 und 16 wird die Schriftgröße von `MeineSchrift` geändert. Das bedeutet, dass sich die Schriftgröße bei allen Elementen ändert, die `MeineSchrift` verwenden.

Im Unterschied dazu wird in dem nächsten Beispiel die Schriftgröße nur für die Buttons geändert.

Listing 54.13: Schrift wird nur für die Buttons geändert (Beispiel486.tcl)

```
1  #!/usr/bin/env wish
2
3  font create MeineSchrift -family Helvetica -size 12 -slant
       roman -weight normal
4  font create MeineSchriftKlein -family Helvetica -size 8
       -slant roman -weight normal
5  font create MeineSchriftGross -family Helvetica -size 16
       -slant roman -weight normal
6
7  ttk::style theme use clam
8
9  ttk::style configure MeinStil.TLabel -font MeineSchrift
10 ttk::style configure MeinStil.TButton -font MeineSchrift
11
12 ttk::label .lb -text "Label" -style MeinStil.TLabel
13 ttk::button .bt -text "Button" -style MeinStil.TButton
14 pack .lb -side top
15 pack .bt -side top
16
17 ttk::button .btKlein -text "Schrift 8" -command {
       ttk::style configure MeinStil.TButton -font
       MeineSchriftKlein}
18 ttk::button .btGross -text "Schrift 16" -command {
       ttk::style configure MeinStil.TButton -font
       MeineSchriftGross}
19 pack .btKlein -side left
20 pack .btGross -side left
```

Klick auf den Button `Schrift 8`:

Klick auf den Button `Schrift 20`:

In den Zeilen 17 und 18 wird die Schrift nur für die Buttons geändert. Alle anderen Elemente behalten ihre Schrift.

54.3.5 Entry

Listing 54.14: Entry (Beispiel487.tcl)

```
1  #!/usr/bin/env wish
2
3  set Farbe1 #0058A2 ; # dunkelblau
4  set Farbe2 #68A2D4 ; # mittelblau
5  set Farbe3 #DEE8F7 ; # hellblau
6  set Farbe4 #FFFFFF ; # weiss
7  set Farbe6 #D0D0D0 ; # hellgrau
8  set Farbe7 #A0A0A0 ; # mittelgrau
9  set Farbe8 #707070 ; # dunkelgrau
10 set Farbe10 #000000 ; # schwarz
11
12 font create MeineSchrift -family Helvetica -size 12 -slant
       roman -weight normal
13
14 ttk::style theme use clam
15
16 ttk::style configure MeinStil.TEntry -fieldbackground
       $Farbe2 ; # Hintergrundfarbe
17 ttk::style configure MeinStil.TEntry -foreground $Farbe1
       ;# Textfarbe
18 ttk::style configure MeinStil.TEntry -background $Farbe1 ;
       # Ecken der Umrandung. Nur verfuegbar, wenn das Thema
       clam (siehe Zeile 14) gewaehlt ist.
19 ttk::style configure MeinStil.TEntry -bordercolor $Farbe1
       ; # Umrandung. Nur verfuegbar, wenn das Thema clam (
       siehe Zeile 14) gewaehlt ist.
```

```
20 ttk::style configure MeinStil.TEntry -lightcolor $Farbe2 ;
     # Der Bereich zwischen der Umrandung und dem
   Eingabebereich. Nur verfuegbar, wenn das Thema clam (
   siehe Zeile 14) gewaehlt ist.
21
22 ttk::style configure MeinStil.TEntry -insertcolor $Farbe1
   ; # Cursorfarbe
23 ttk::style configure MeinStil.TEntry -selectbackground
   $Farbe1 ; # Hintergrundfarbe des selektierten Bereichs
24 ttk::style configure MeinStil.TEntry -selectforeground
   $Farbe2 ; # Textfarbe des selektierten Bereichs
25
26 ttk::style map MeinStil.TEntry -fieldbackground [list
   active $Farbe2 disabled $Farbe6 readonly $Farbe4]
27 ttk::style map MeinStil.TEntry -foreground [list active
   $Farbe1 disabled $Farbe8 readonly $Farbe10]
28 ttk::style map MeinStil.TEntry -background [list active
   $Farbe1 disabled $Farbe8 readonly $Farbe10] ; # Nur
   verfuegbar, wenn das Thema clam (siehe Zeile 14)
   gewaehlt ist.
29 ttk::style map MeinStil.TEntry -bordercolor [list active
   $Farbe1 disabled $Farbe8 readonly $Farbe10] ; # Nur
   verfuegbar, wenn das Thema clam (siehe Zeile 14)
   gewaehlt ist.
30 ttk::style map MeinStil.TEntry -lightcolor [list active
   $Farbe1 disabled $Farbe8 readonly $Farbe10] ; # Nur
   verfuegbar, wenn das Thema clam (siehe Zeile 14)
   gewaehlt ist.
31 ttk::style map MeinStil.TEntry -selectbackground [list
   active $Farbe1 readonly $Farbe10]
32 ttk::style map MeinStil.TEntry -selectforeground [list
   active $Farbe2 readonly $Farbe4]
33
34 set Eingabe "Text"
35 ttk::entry .en -textvariable Eingabe -style
   MeinStil.TEntry -font MeineSchrift
36 pack .en -side left
37
38 ttk::button .btKlein -text "Schrift 8" -command {font
   configure MeineSchrift -size 8; append Eingabe ""}
39 ttk::button .btGross -text "Schrift 16" -command {font
   configure MeineSchrift -size 16; append Eingabe ""}
40 pack .btKlein -side top
41 pack .btGross -side top
42
43 ttk::button .btNormal -text "Normal" -command {.en
   configure -state normal}
44 ttk::button .btActive -text "Active" -command {.en
   configure -state active}
45 ttk::button .btDisabled -text "Disabled" -command {.en
   configure -state disabled}
46 ttk::button .btReadonly -text "Readonly" -command {.en
   configure -state readonly}
```

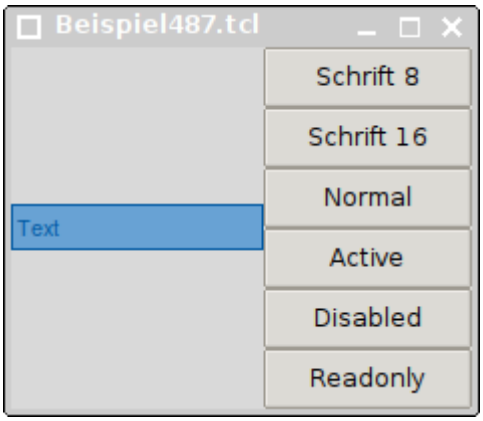
```
47 pack .btNormal -side top
48 pack .btActive -side top
49 pack .btDisabled -side top
50 pack .btReadonly -side top
```

Einstellung der Schrift: Zeilen 12, 35, 38 und 39.

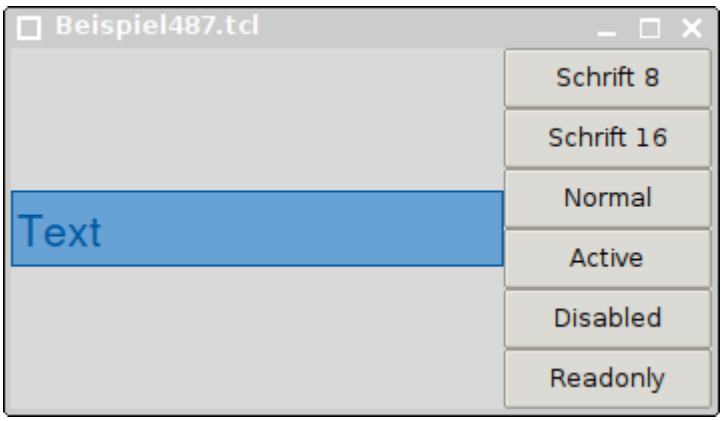
Beim `entry`-Element muss im Unterschied zu den anderen ttk-Elementen die Schrift über die Option `-font` zugeordnet werden (Zeile 33). Es reicht nicht aus, nur den Stil zuzuordnen. Wenn die Schriftgröße geändert wird (Zeilen 36 und 37) muss man zusätzlich eine Veränderung des Textes simulieren, damit der Text korrekt (d. h. passend zur neuen Schriftgröße) innerhalb des `entry`-Elements positioniert wird. Das erreicht man z. B. durch den `append`-Befehl.

Nach dem Programmstart ist das `entry`-Element im Zustand `normal` und hat Schriftgröße 12:

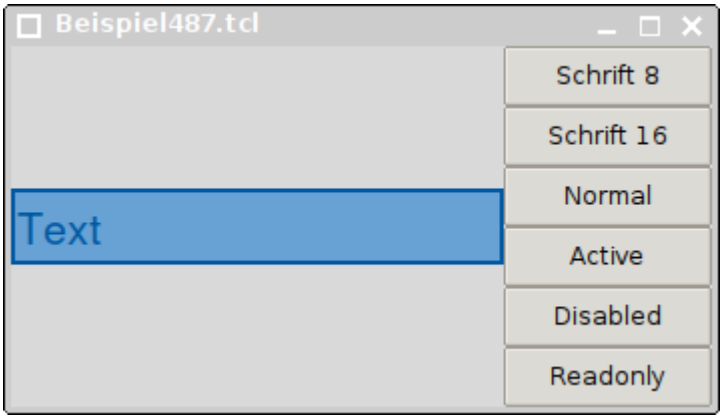
Klick auf den Button `Schrift 8`:



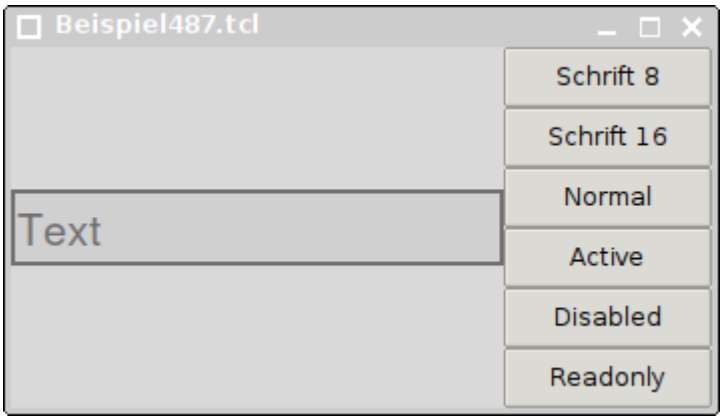
Klick auf den Button `Schrift 16`:



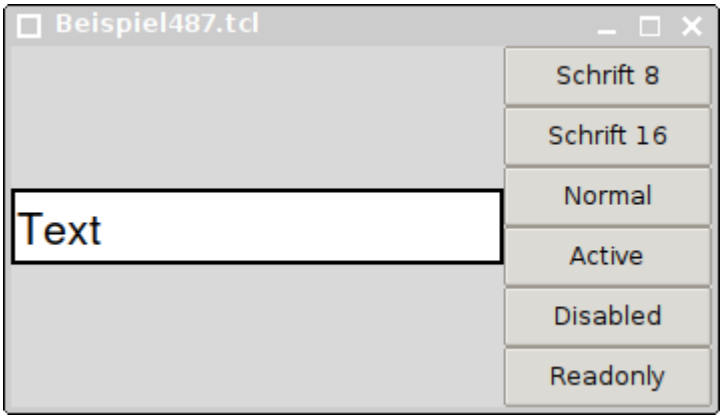
Klick auf den Button `Active`:



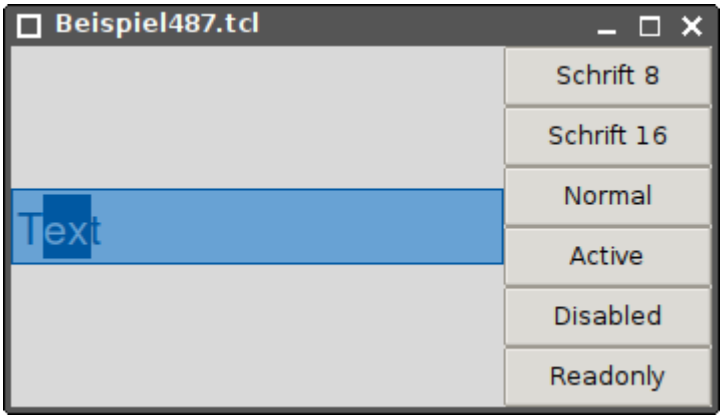
Klick auf den Button `Disabled`:



Klick auf den Button `Readonly`:



Klick auf den Button `Normal` und selektierter Text:



54.3.6 Labelframe

Die Formatierung eines `labelframe`-Elements ist in Tcl/Tk nicht vollständig realisiert. Wenn man das Thema `clam` verwendet, wird die Beschriftung des Labelframes nicht auf der Umrandung platziert, sondern darüber. Verwendet man stattdessen das `default`-Thema wird die Beschriftung richtig platziert, aber dann ist die Rahmenfarbe immer schwarz und kann nicht geändert werden.

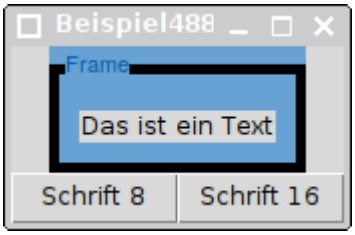
Listing 54.15: Labelframe mit default-Thema (Beispiel488.tcl)

```
#!/usr/bin/env wish

set Farbe1 #0058A2 ; # dunkelblau
set Farbe2 #68A2D4 ; # mittelblau
set Farbe3 #DEE8F7 ; # hellblau
set Farbe4 #FFFFFF ; # weiss
set Farbe6 #D0D0D0 ; # hellgrau
set Farbe7 #A0A0A0 ; # mittelgrau
set Farbe8 #707070 ; # dunkelgrau
set Farbe10 #000000 ; # schwarz

font create MeineSchrift -family Helvetica -size 12 -slant
    roman -weight normal
```

```
13
14 ttk::style theme use default
15
16 ttk::style configure MeinStil.TLabelframe -background $Farbe2
17 ttk::style configure MeinStil.TLabelframe -bordercolor $Farbe1 ; # nur verfuegbar, wenn das Thema clam (siehe Zeile 8) gewaehlt ist
18 ttk::style configure MeinStil.TLabelframe -font MeineSchrift
19
20 ttk::style configure MeinStil.TLabelframe.Label -background $Farbe2
21 ttk::style configure MeinStil.TLabelframe.Label -foreground $Farbe1
22 ttk::style configure MeinStil.TLabelframe.Label -font MeineSchrift
23
24 ttk::label .lbFuerLabelframe -text "Frame" -style MeinStil.TLabelframe.Label
25 ttk::labelframe .fr -padding {10 10 10 10} -labelwidget .lbFuerLabelframe -labelanchor nw -borderwidth 5 -relief solid -style MeinStil.TLabelframe
26 ttk::label .fr.lb -text "Das ist ein Text"
27 pack .fr.lb -side top
28 pack .fr
29
30 ttk::button .btKlein -text "Schrift 8" -command {font configure MeineSchrift -size 8}
31 ttk::button .btGross -text "Schrift 16" -command {font configure MeineSchrift -size 16}
32 pack .btKlein -side left
33 pack .btGross -side right
```

Einstellung der Schrift: Zeilen 12, 18, 22, 30 und 31.

Klick auf den Button `Schrift 8`:



Klick auf den Button `Schrift 16`:

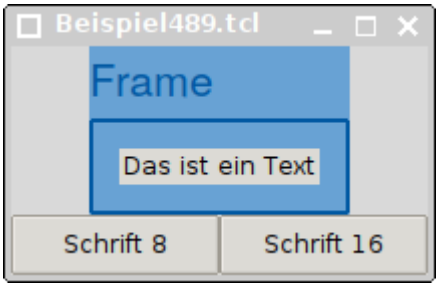
Listing 54.16: Labelframe mit clam-Thema (Beispiel489.tcl)

```
1  #!/usr/bin/env wish
2
3  set Farbe1 #0058A2 ; # dunkelblau
4  set Farbe2 #68A2D4 ; # mittelblau
5  set Farbe3 #DEE8F7 ; # hellblau
6  set Farbe4 #FFFFFF ; # weiss
7  set Farbe6 #D0D0D0 ; # hellgrau
8  set Farbe7 #A0A0A0 ; # mittelgrau
9  set Farbe8 #707070 ; # dunkelgrau
10 set Farbe10 #000000 ; # schwarz
11
12 font create MeineSchrift -family Helvetica -size 12 -slant ↩
       roman -weight normal
13
14 ttk::style theme use clam
15
16 ttk::style configure MeinStil.TLabelframe -background ↩
       $Farbe2
17 ttk::style configure MeinStil.TLabelframe -bordercolor ↩
       $Farbe1 ; # nur verfuegbar, wenn das Thema clam (siehe ↩
       Zeile 8) gewaehlt ist
18 ttk::style configure MeinStil.TLabelframe -font ↩
       MeineSchrift
19
20 ttk::style configure MeinStil.TLabelframe.Label ↩
       -background $Farbe2
21 ttk::style configure MeinStil.TLabelframe.Label ↩
       -foreground $Farbe1
22 ttk::style configure MeinStil.TLabelframe.Label -font ↩
       MeineSchrift
23
24 ttk::label .lbFuerLabelframe -text "Frame" -style ↩
       MeinStil.TLabelframe.Label
25 ttk::labelframe .fr -padding {10 10 10 10} -labelwidget ↩
       .lbFuerLabelframe -labelanchor nw -borderwidth 5 ↩
       -relief solid -style MeinStil.TLabelframe
26 ttk::label .fr.lb -text "Das ist ein Text"
27 pack .fr.lb -side top
28 pack .fr
29
30 ttk::button .btKlein -text "Schrift 8" -command {font ↩
       configure MeineSchrift -size 8}
```

```
31 ttk::button .btGross -text "Schrift 16" -command {font configure MeineSchrift -size 16}
32 pack .btKlein -side left
33 pack .btGross -side right
```

Einstellung der Schrift: Zeilen 12, 18, 22, 30 und 31.
Klick auf den Button `Schrift 8`:

Klick auf den Button `Schrift 16`:



54.3.7 Checkbutton

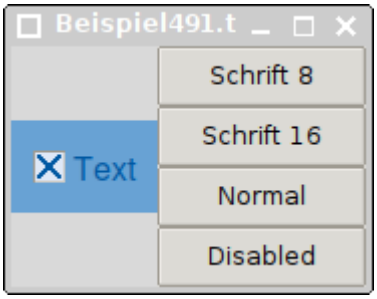
Listing 54.17: Checkbutton (Beispiel491.tcl)

```
1  #!/usr/bin/env wish
2
3  set Farbe1 #0058A2 ; # dunkelblau
4  set Farbe2 #68A2D4 ; # mittelblau
5  set Farbe3 #DEE8F7 ; # hellblau
6  set Farbe4 #FFFFFF ; # weiss
7  set Farbe6 #D0D0D0 ; # hellgrau
8  set Farbe7 #A0A0A0 ; # mittelgrau
9  set Farbe8 #707070 ; # dunkelgrau
10 set Farbe10 #000000 ; # schwarz
11
12 font create MeineSchrift -family Helvetica -size 12 -slant roman -weight normal
13
14 ttk::style theme use clam
15
16 ttk::style configure MeinStil.TCheckbutton -background $Farbe2
17 ttk::style configure MeinStil.TCheckbutton -foreground $Farbe1
```

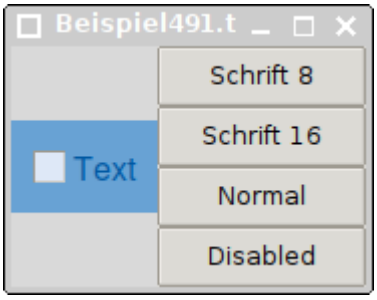
```
18 ttk::style configure MeinStil.TCheckbutton ↩
     -indicatorbackground $Farbe3
19 ttk::style configure MeinStil.TCheckbutton ↩
     -indicatorforeground $Farbe1
20 ttk::style configure MeinStil.TCheckbutton -indicatorsize ↩
     15
21 ttk::style configure MeinStil.TCheckbutton -font ↩
     MeineSchrift
22
23 ttk::style map MeinStil.TCheckbutton -background [list ↩
     active $Farbe2 disabled $Farbe7]
24 ttk::style map MeinStil.TCheckbutton -foreground [list ↩
     active $Farbe1 disabled $Farbe8]
25 ttk::style map MeinStil.TCheckbutton -indicatorbackground ↩
     [list active $Farbe3 disabled $Farbe6]
26 ttk::style map MeinStil.TCheckbutton -indicatorforeground ↩
     [list active $Farbe1 disabled $Farbe8]
27
28 set AnAus 1
29 ttk::checkbutton .cb -text "Text" -variable AnAus -padding↩
      {10 10 10 10} -style MeinStil.TCheckbutton
30 pack .cb -side left
31
32 ttk::button .btKlein -text "Schrift 8" -command {
33   font configure MeineSchrift -size 8
34   ttk::style configure MeinStil.TCheckbutton ↩
       -indicatorsize 10
35 }
36 ttk::button .btGross -text "Schrift 16" -command {
37   font configure MeineSchrift -size 16
38   ttk::style configure MeinStil.TCheckbutton ↩
       -indicatorsize 20
39 }
40 pack .btKlein -side top
41 pack .btGross -side top
42
43 ttk::button .btNormal -text "Normal" -command {.cb ↩
     configure -state normal}
44 ttk::button .btDisabled -text "Disabled" -command {.cb ↩
     configure -state disabled}
45 pack .btNormal -side top
46 pack .btDisabled -side top
```

Einstellung der Schrift: Zeilen 12, 21, 33 und 37.

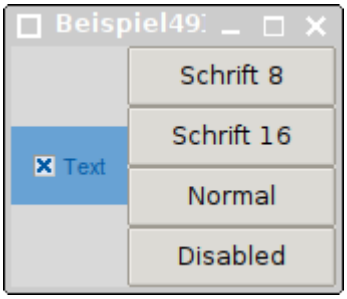
Wenn das Kästchen markiert ist:



Wenn das Kästchen nicht markiert ist:



Klick auf den Button `Schrift 8`:



Klick auf den Button `Schrift 16`:

Klick auf den Button `Disabled`:

54.3.8 Radiobutton

In den Zeilen 16 bis 19 wird ein Rahmen für die Hintergrundfarbe erstellt. Ohne diesen Rahmen hätten die Radiobuttons mit kurzem Text am rechten Rand eine andere Hintergrundfarbe als die Radiobuttons mit langem Text.

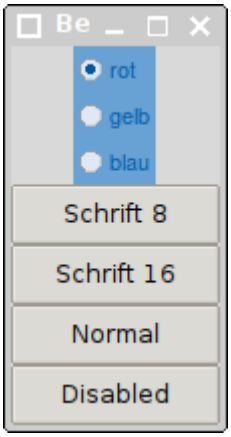
Listing 54.18: Radiobutton (Beispiel493.tcl)

```
1  #!/usr/bin/env wish
2
3  set Farbe1 #0058A2 ; # dunkelblau
4  set Farbe2 #68A2D4 ; # mittelblau
5  set Farbe3 #DEE8F7 ; # hellblau
6  set Farbe4 #FFFFFF ; # weiss
7  set Farbe6 #D0D0D0 ; # hellgrau
8  set Farbe7 #A0A0A0 ; # mittelgrau
9  set Farbe8 #707070 ; # dunkelgrau
10 set Farbe10 #000000 ; # schwarz
11
12 font create MeineSchrift -family Helvetica -size 12 -slant
       roman -weight normal
13
14 ttk::style theme use clam
15
16 # Ein frame-Element, um die Hintergrundfarbe zu setzen
17 ttk::style configure MeinStil.TFrame -background $Farbe2
18 ttk::frame .fr -style MeinStil.TFrame
19 pack .fr
20
21 ttk::style configure MeinStil.TRadiobutton -background
       $Farbe2
22 ttk::style configure MeinStil.TRadiobutton -foreground
       $Farbe1
23 ttk::style configure MeinStil.TRadiobutton
       -indicatorbackground $Farbe3
24 ttk::style configure MeinStil.TRadiobutton
       -indicatorforeground $Farbe1
25 ttk::style configure MeinStil.TRadiobutton -indicatorsize
       15
26 ttk::style configure MeinStil.TRadiobutton -font
       MeineSchrift
27
```

```
28 ttk::style map MeinStil.TRadiobutton -background [list ↩
     active $Farbe2 disabled $Farbe7]
29 ttk::style map MeinStil.TRadiobutton -foreground [list ↩
     active $Farbe1 disabled $Farbe8]
30 ttk::style map MeinStil.TRadiobutton -indicatorbackground ↩
     [list active $Farbe3 disabled $Farbe6]
31 ttk::style map MeinStil.TRadiobutton -indicatorforeground ↩
     [list active $Farbe1 disabled $Farbe8]
32
33 set Farbe "rot"
34 ttk::radiobutton .fr.rbRot -text "rot" -variable Farbe ↩
     -value "rot" -style MeinStil.TRadiobutton
35 ttk::radiobutton .fr.rbGelb -text "gelb" -variable Farbe ↩
     -value "gelb" -style MeinStil.TRadiobutton
36 ttk::radiobutton .fr.rbBlau -text "blau" -variable Farbe ↩
     -value "blau" -style MeinStil.TRadiobutton
37
38 pack .fr.rbRot -side top -anchor w
39 pack .fr.rbGelb -side top -anchor w
40 pack .fr.rbBlau -side top -anchor w
41
42 ttk::button .btKlein -text "Schrift 8" -command {
43   font configure MeineSchrift -size 8
44   ttk::style configure MeinStil.TRadiobutton ↩
       -indicatorsize 10
45 }
46 ttk::button .btGross -text "Schrift 16" -command {
47   font configure MeineSchrift -size 16
48   ttk::style configure MeinStil.TRadiobutton ↩
       -indicatorsize 20
49 }
50 pack .btKlein -side top
51 pack .btGross -side top
52
53 ttk::button .btNormal -text "Normal" -command {
54         .fr.rbRot configure -state normal
55         .fr.rbGelb configure -state normal
56         .fr.rbBlau configure -state normal
57         ttk::style configure MeinStil.TFrame -background ↩
             $Farbe2
58 }
59 ttk::button .btDisabled -text "Disabled" -command {
60         .fr.rbRot configure -state disabled
61         .fr.rbGelb configure -state disabled
62         .fr.rbBlau configure -state disabled
63         ttk::style configure MeinStil.TFrame -background ↩
             $Farbe7
64 }
65 pack .btNormal -side top
66 pack .btDisabled -side top
```

Einstellung der Schrift: Zeilen 12, 26, 43 und 47.
Nach dem Programmstart:

Klick auf den Button `Schrift 8`:



Klick auf den Button `Schrift 16`:

Klick auf den Button `Disabled`:

54.3.9 Menubutton

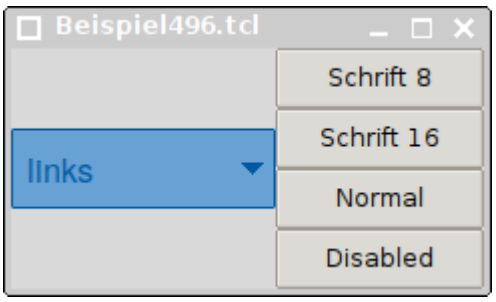
Listing 54.19: Menubutton (Beispiel496.tcl)

```
#!/usr/bin/env wish

set Farbe1 #0058A2 ; # dunkelblau
set Farbe2 #68A2D4 ; # mittelblau
set Farbe3 #DEE8F7 ; # hellblau
set Farbe4 #FFFFFF ; # weiss
set Farbe6 #D0D0D0 ; # hellgrau
set Farbe7 #A0A0A0 ; # mittelgrau
set Farbe8 #707070 ; # dunkelgrau
set Farbe10 #000000 ; # schwarz

font create MeineSchrift -family Helvetica -size 12 -slant roman -weight normal

ttk::style theme use clam

ttk::style configure MeinStil.TMenubutton -background $Farbe2
ttk::style configure MeinStil.TMenubutton -foreground $Farbe1
ttk::style configure MeinStil.TMenubutton -arrowcolor $Farbe1
ttk::style configure MeinStil.TMenubutton -bordercolor $Farbe1
ttk::style configure MeinStil.TMenubutton -lightcolor $Farbe2
ttk::style configure MeinStil.TMenubutton -darkcolor $Farbe2
ttk::style configure MeinStil.TMenubutton -font MeineSchrift

```

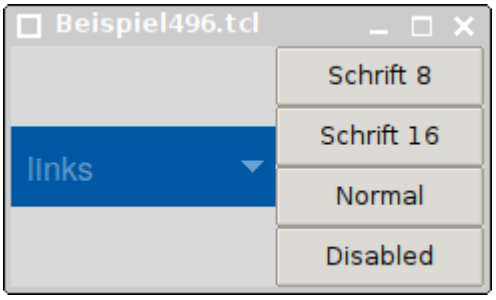
```
24 ttk::style map MeinStil.TMenubutton -background [list ↩
      active $Farbe1 disabled $Farbe7]
25 ttk::style map MeinStil.TMenubutton -foreground [list ↩
      active $Farbe2 disabled $Farbe8]
26 ttk::style map MeinStil.TMenubutton -bordercolor [list ↩
      active $Farbe1 disabled $Farbe7]
27 ttk::style map MeinStil.TMenubutton -lightcolor [list ↩
      active $Farbe1 disabled $Farbe7]
28 ttk::style map MeinStil.TMenubutton -darkcolor [list ↩
      active $Farbe1 disabled $Farbe7]
29 ttk::style map MeinStil.TMenubutton -arrowcolor [list ↩
      active $Farbe2 disabled $Farbe8]
30
31 ttk::style configure MeinStil.TMenubutton -arrowsize 8
32
33 set Ausrichtung "links"
34 ttk::menubutton .mb -textvariable Ausrichtung -style ↩
      MeinStil.TMenubutton
35 menu .mb.menu -tearoff 0 -font MeineSchrift -background ↩
      $Farbe2 -foreground $Farbe1 -activebackground $Farbe1 ↩
      -activeforeground $Farbe2 -disabledforeground $Farbe8 ↩
      -selectcolor $Farbe1
36 .mb configure -menu .mb.menu
37 .mb.menu add radiobutton -value "links" -variable ↩
      Ausrichtung -label "links"
38 .mb.menu add radiobutton -value "zentriert" -variable ↩
      Ausrichtung -label "zentriert" -state disabled
39 .mb.menu add radiobutton -value "rechts" -variable ↩
      Ausrichtung -label "rechts"
40 .mb.menu add cascade -label "Sonstiges" -menu ↩
      .mb.menu.sonstiges
41 menu .mb.menu.sonstiges -tearoff 0 -font MeineSchrift ↩
      -background $Farbe2 -foreground $Farbe1 ↩
      -activebackground $Farbe1 -activeforeground $Farbe2 ↩
      -disabledforeground $Farbe8 -selectcolor $Farbe1
42 .mb.menu.sonstiges add radiobutton -value "Blocksatz" ↩
      -variable Ausrichtung -label "Blocksatz"
43 .mb.menu.sonstiges add radiobutton -value "Sonstiges" ↩
      -variable Ausrichtung -label "Sonstiges"
44
45 pack .mb -side left
46
47 ttk::button .btKlein -text "Schrift 8" -command {font ↩
      configure MeineSchrift -size 8; ttk::style configure ↩
      MeinStil.TMenubutton -arrowsize 4}
48 ttk::button .btGross -text "Schrift 16" -command {font ↩
      configure MeineSchrift -size 16; ttk::style configure ↩
      MeinStil.TMenubutton -arrowsize 10}
49 pack .btKlein -side top
50 pack .btGross -side top
51
52 ttk::button .btNormal -text "Normal" -command {.mb ↩
      configure -state normal}
```

```
53 ttk::button .btDisabled -text "Disabled" -command {.mb ↩
      configure -state disabled}
54 pack .btNormal -side top
55 pack .btDisabled -side top
```

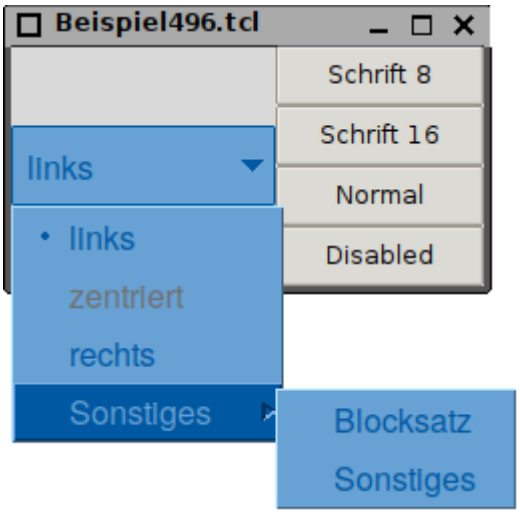
Einstellung der Schrift: Zeilen 12, 22, 35, 41, 47 und 48.
Nach dem Programmstart:



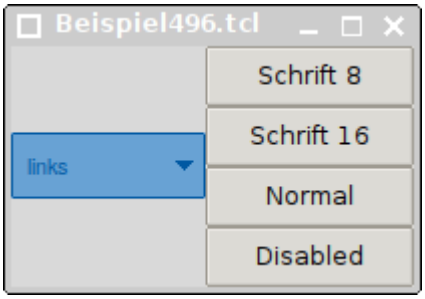
Wenn die Maus über dem Menubutton ist:



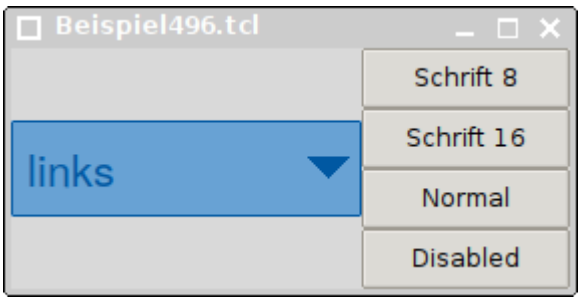
Aufgeklapptes Menu:



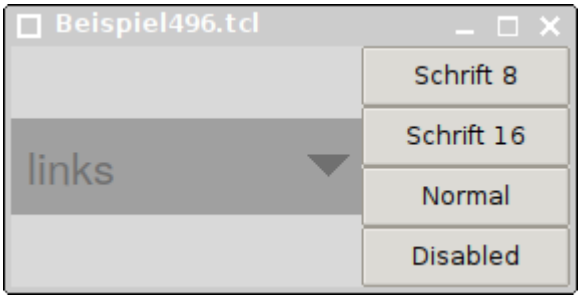
Klick auf den Button `Schrift 8`:



Klick auf den Button `Schrift 16`:



Klick auf den Button `Disabled`:



54.3.10 Spinbox

Listing 54.20: Spinbox (Beispiel497.tcl)

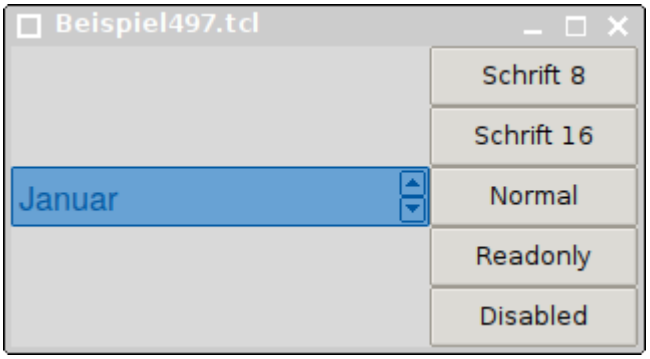
```
#!/usr/bin/env wish

set Farbe1 #0058A2 ; # dunkelblau
set Farbe2 #68A2D4 ; # mittelblau
set Farbe3 #DEE8F7 ; # hellblau
set Farbe4 #FFFFFF ; # weiss
set Farbe6 #D0D0D0 ; # hellgrau
set Farbe7 #A0A0A0 ; # mittelgrau
set Farbe8 #707070 ; # dunkelgrau
set Farbe10 #000000 ; # schwarz

font create MeineSchrift -family Helvetica -size 12 -slant roman -weight normal

```

```
14 ttk::style theme use clam
15
16 ttk::style configure MeinStil.TSpinbox -fieldbackground ↩
      $Farbe2
17 ttk::style configure MeinStil.TSpinbox -bordercolor ↩
      $Farbe1
18 ttk::style configure MeinStil.TSpinbox -lightcolor $Farbe2
19 ttk::style configure MeinStil.TSpinbox -darkcolor $Farbe2
20 ttk::style configure MeinStil.TSpinbox -arrowsize 13
21 ttk::style configure MeinStil.TSpinbox -font MeineSchrift
22
23 ttk::style configure MeinStil.TSpinbox -background $Farbe2
24 ttk::style configure MeinStil.TSpinbox -foreground $Farbe1
25 ttk::style configure MeinStil.TSpinbox -arrowcolor $Farbe1
26
27 ttk::style configure MeinStil.TSpinbox -insertcolor ↩
      $Farbe1 ; # Cursorfarbe
28 ttk::style configure MeinStil.TSpinbox -selectbackground ↩
      $Farbe1
29 ttk::style configure MeinStil.TSpinbox -selectforeground ↩
      $Farbe2
30
31 ttk::style map MeinStil.TSpinbox -background [list active ↩
      $Farbe2 disabled $Farbe7 readonly $Farbe4]
32 ttk::style map MeinStil.TSpinbox -foreground [list active ↩
      $Farbe2 disabled $Farbe8 readonly $Farbe10]
33 ttk::style map MeinStil.TSpinbox -bordercolor [list active↩
       $Farbe2 disabled $Farbe8 readonly $Farbe4]
34 ttk::style map MeinStil.TSpinbox -lightcolor [list active ↩
      $Farbe1 disabled $Farbe7 readonly $Farbe10]
35 ttk::style map MeinStil.TSpinbox -darkcolor [list active ↩
      $Farbe1 disabled $Farbe7 readonly $Farbe10]
36 ttk::style map MeinStil.TSpinbox -arrowcolor [list active ↩
      $Farbe1 disabled $Farbe8 readonly $Farbe10]
37 ttk::style map MeinStil.TSpinbox -fieldbackground [list ↩
      active $Farbe2 disabled $Farbe7 readonly $Farbe4]
38
39 ttk::style configure Disabled.MeinStil.TSpinbox
40 ttk::style map Disabled.MeinStil.TSpinbox -background [↩
      list active $Farbe7 disabled $Farbe7]
41 ttk::style map Disabled.MeinStil.TSpinbox -foreground [↩
      list active $Farbe8 disabled $Farbe8]
42 ttk::style map Disabled.MeinStil.TSpinbox -bordercolor [↩
      list active $Farbe8 disabled $Farbe8]
43 ttk::style map Disabled.MeinStil.TSpinbox -lightcolor [↩
      list active $Farbe7 disabled $Farbe7]
44 ttk::style map Disabled.MeinStil.TSpinbox -darkcolor [list↩
       active $Farbe7 disabled $Farbe7]
45 ttk::style map Disabled.MeinStil.TSpinbox -arrowcolor [↩
      list active $Farbe8 disabled $Farbe8]
46 ttk::style map Disabled.MeinStil.TSpinbox -fieldbackground↩
       [list active $Farbe7 disabled $Farbe7]
47
```

```
48 ttk::style configure Readonly.MeinStil.TSpinbox
49 ttk::style configure Readonly.MeinStil.TSpinbox ↲
     -insertcolor $Farbe4
50 ttk::style configure Readonly.MeinStil.TSpinbox ↲
     -selectbackground $Farbe4
51 ttk::style configure Readonly.MeinStil.TSpinbox ↲
     -selectforeground $Farbe10
52 ttk::style map Readonly.MeinStil.TSpinbox -background [↲
     list active $Farbe4 readonly $Farbe4]
53 ttk::style map Readonly.MeinStil.TSpinbox -foreground [↲
     list active $Farbe10 readonly $Farbe10]
54 ttk::style map Readonly.MeinStil.TSpinbox -bordercolor [↲
     list active $Farbe4 readonly $Farbe4]
55 ttk::style map Readonly.MeinStil.TSpinbox -lightcolor [↲
     list active $Farbe10 readonly $Farbe10]
56 ttk::style map Readonly.MeinStil.TSpinbox -darkcolor [list↲
      active $Farbe10 readonly $Farbe10]
57 ttk::style map Readonly.MeinStil.TSpinbox -arrowcolor [↲
     list active $Farbe10 readonly $Farbe10]
58 ttk::style map Readonly.MeinStil.TSpinbox -fieldbackground↲
      [list active $Farbe4 readonly $Farbe4]
59
60 set Liste {Januar Februar Maerz April Mai Juni}
61 set Monat [lindex $Liste 0]
62 ttk::spinbox .sb -values $Liste -textvariable Monat -state↲
      normal -style MeinStil.TSpinbox -font MeineSchrift
63 pack .sb -side left
64
65 ttk::button .btKlein -text "Schrift 8" -command {font ↲
     configure MeineSchrift -size 8; ttk::style configure ↲
     MeinStil.TSpinbox -arrowsize 11; set Monat $Monat}
66 ttk::button .btGross -text "Schrift 16" -command {font ↲
     configure MeineSchrift -size 16; ttk::style configure ↲
     MeinStil.TSpinbox -arrowsize 15; set Monat $Monat}
67 pack .btKlein -side top
68 pack .btGross -side top
69
70 ttk::button .btNormal -text "Normal" -command {.sb ↲
     configure -state normal; .sb configure -style ↲
     MeinStil.TSpinbox}
71 ttk::button .btReadonly -text "Readonly" -command {.sb ↲
     configure -state readonly; .sb configure -style ↲
     Readonly.MeinStil.TSpinbox}
72 ttk::button .btDisabled -text "Disabled" -command {.sb ↲
     configure -state disabled; .sb configure -style ↲
     Disabled.MeinStil.TSpinbox}
73 pack .btNormal -side top
74 pack .btReadonly -side top
75 pack .btDisabled -side top
```

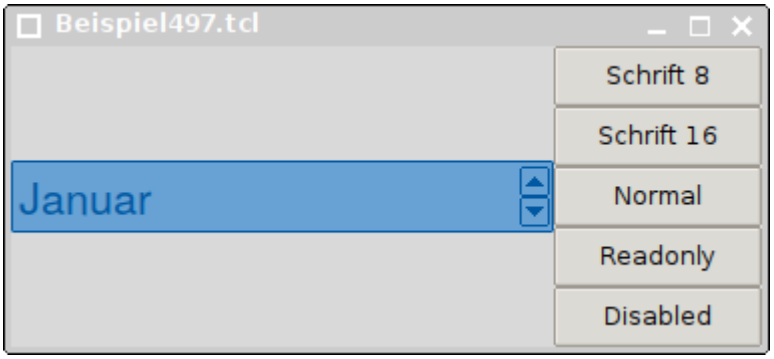
Einstellung der Schrift: Zeilen 12, 21, 62, 65 und 66.

Nach dem Programmstart ist das Programm im Status `normal`:



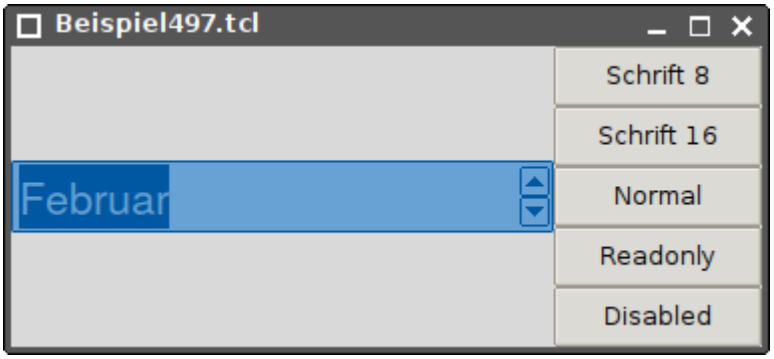
Klick auf den Button `Schrift 8`:

Klick auf den Button `Schrift 16`:

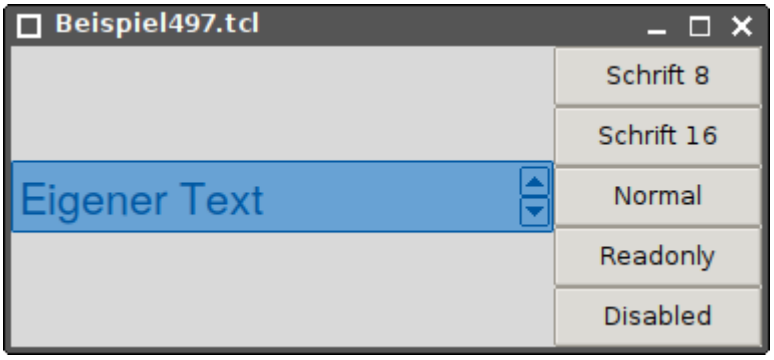


Damit der Text bei Wahl einer anderen Schriftgröße innerhalb der Spinbox richtig platziert wird, muss die scheinbar sinnlose Anweisung `set Monat $Monat` ausgeführt werden (Zeilen 65 und 66).

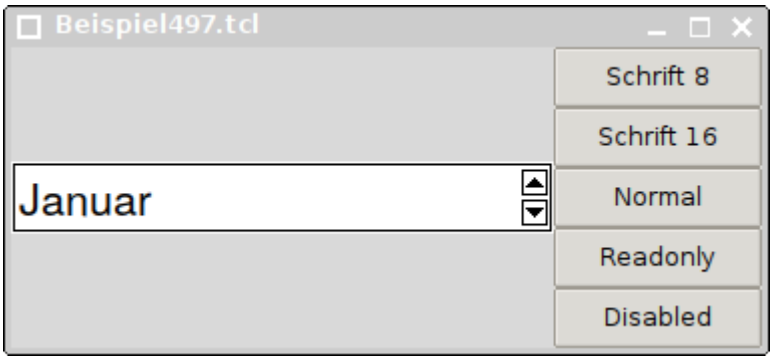
Klick auf einen Auswahlpfeil:



Eingabe eines eigenen Texts:

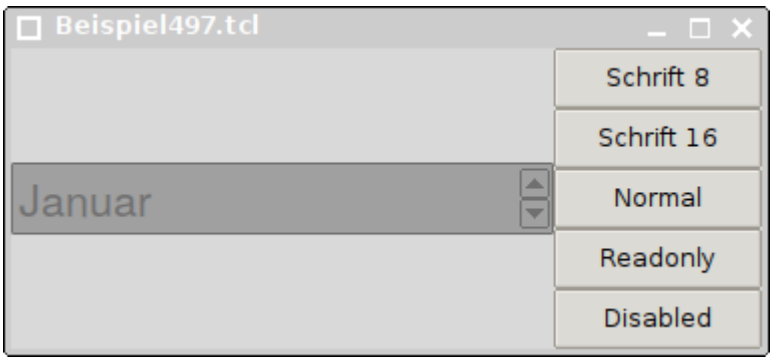


Klick auf den Button `Readonly`:



In den Zeilen 48 bis 58 wird ein eigener Stil für den Readonly-Zustand definiert. Er basiert auf dem Stil `MeinStil.TSpinbox` (Zeile 48) und hat dann ein paar abweichende Farben (Zeilen 49 bis 58).

Klick auf den Button `Disabled`:



In den Zeilen 39 bis 46 wird ein eigener Stil für den Disabled-Zustand definiert. Er basiert auf dem Stil `MeinStil.TSpinbox` (Zeile 39) und hat dann ein paar abweichende Farben (Zeilen 40 bis 46).

54.3.11 Listbox

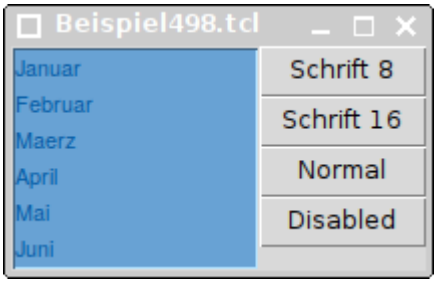
Die Listbox gibt es nur als klassisches Element, so dass man nicht den Befehl `ttk::style` verwenden kann.

Listing 54.21: Listbox (Beispiel498.tcl)

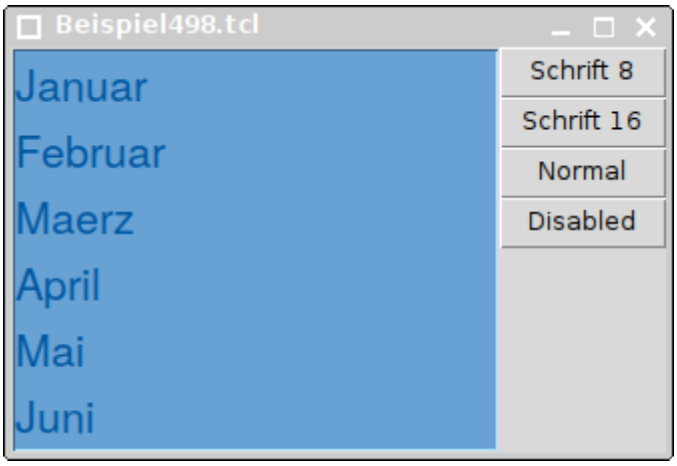
```
1  #!/usr/bin/env wish
2
3  set Farbe1 #0058A2 ; # dunkelblau
4  set Farbe2 #68A2D4 ; # mittelblau
5  set Farbe3 #DEE8F7 ; # hellblau
6  set Farbe4 #FFFFFF ; # weiss
7  set Farbe6 #D0D0D0 ; # hellgrau
8  set Farbe7 #A0A0A0 ; # mittelgrau
9  set Farbe8 #707070 ; # dunkelgrau
10 set Farbe10 #000000 ; # schwarz
11
12 font create MeineSchrift -family Helvetica -size 12 -slant
       roman -weight normal
13
14 set Liste {Januar Februar Maerz April Mai Juni}
15 listbox .lb -listvariable Liste -height 6 -font
       MeineSchrift -background $Farbe2 -foreground $Farbe1
       -disabledforeground $Farbe8 -selectbackground $Farbe1
       -selectforeground $Farbe2
16 pack .lb -side left
17
18 ttk::button .btKlein -text "Schrift 8" -command {font
       configure MeineSchrift -size 8}
19 ttk::button .btGross -text "Schrift 16" -command {font
       configure MeineSchrift -size 16}
20 pack .btKlein -side top
21 pack .btGross -side top
22
23 ttk::button .btNormal -text "Normal" -command {.lb
       configure -state normal; .lb configure -background
```

```
    $Farbe2}
24 ttk::button .btDisabled -text "Disabled" -command {.lb ↩
    configure -state disabled; .lb configure -background ↩
    $Farbe7}
25 pack .btNormal -side top
26 pack .btDisabled -side top
```

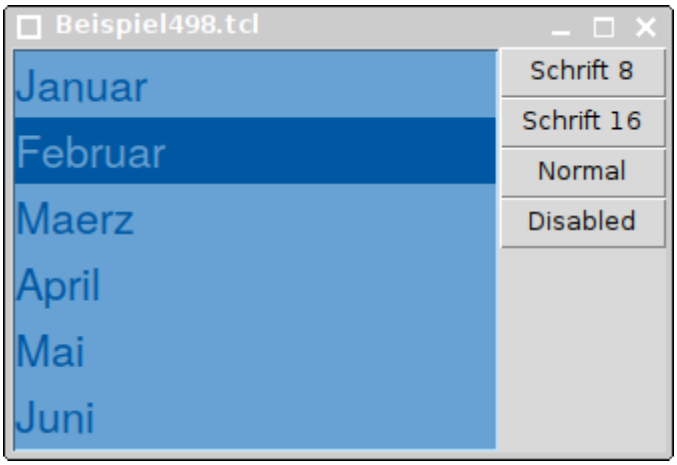
Einstellung der Schrift: Zeilen 12, 15, 18 und 19.
Klick auf den Button `Schrift 8`:



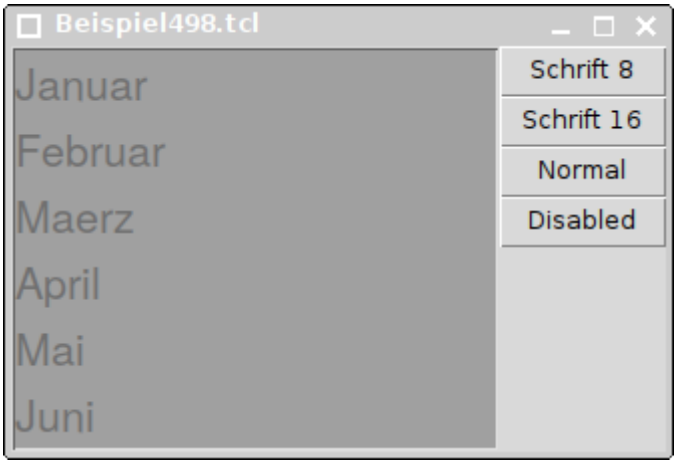
Klick auf den Button `Schrift 16`:



Klick auf einen Eintrag:



Klick auf den Button `Disabled`:



54.3.12 Scrollbar

Im nachstehenden Beispiel wird das Layout der vertikalen (senkrechten) Scroll-Leiste verändert. Die Befehle gelten analog auch für die horizontale Scroll-Leiste. In diesem Fall heißt es dann `Horizontal.TScrollbar` statt `Vertical.TScrollbar`.

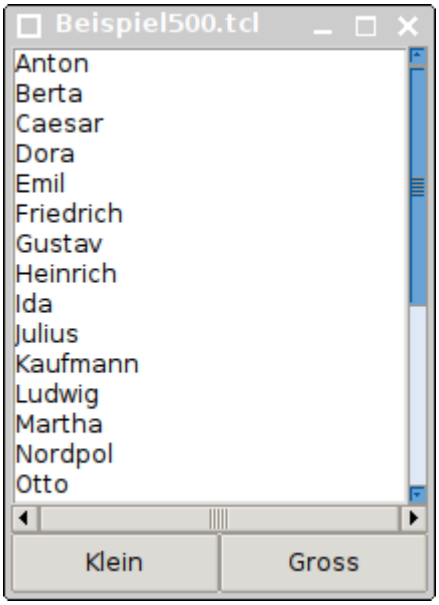
Listing 54.22: Scrollbar (Beispiel500.tcl)

```
#!/usr/bin/env wish

proc StatusPruefen {} {
  uplevel #0 {
    if {[.fr.sbY instate disabled]} {
      ttk::style map MeinStil.Vertical.TScrollbar ⤶
         -background [list active $Farbe7 disabled $Farbe7⤶
         ]
      ttk::style map MeinStil.Vertical.TScrollbar ⤶
         -bordercolor [list active $Farbe8 disabled ⤶
         $Farbe8]
      ttk::style map MeinStil.Vertical.TScrollbar ⤶
         -lightcolor [list active $Farbe7 disabled $Farbe7⤶
         ]
      ttk::style map MeinStil.Vertical.TScrollbar ⤶
         -darkcolor [list active $Farbe8 disabled $Farbe8]
      ttk::style map MeinStil.Vertical.TScrollbar ⤶
         -arrowcolor [list active $Farbe8 disabled $Farbe8⤶
         ]
    } else {
      ttk::style map MeinStil.Vertical.TScrollbar ⤶
         -background [list active $Farbe2 disabled $Farbe7⤶
         ]
      ttk::style map MeinStil.Vertical.TScrollbar ⤶
         -bordercolor [list active $Farbe1 disabled ⤶
         $Farbe8]
      ttk::style map MeinStil.Vertical.TScrollbar ⤶
         -lightcolor [list active $Farbe2 disabled $Farbe7⤶
```

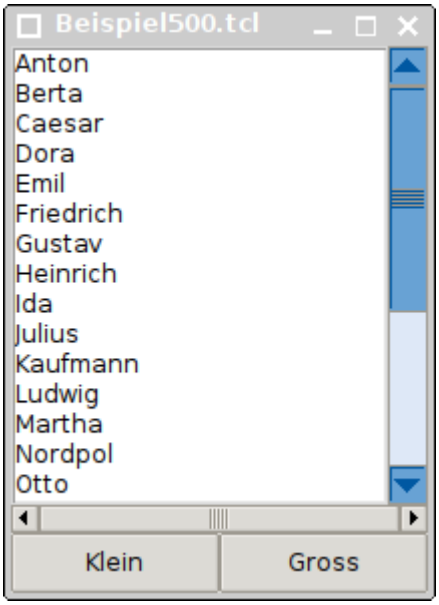
```
            ]
15        ttk::style map MeinStil.Vertical.TScrollbar ↩
            -darkcolor [list active $Farbe2 disabled $Farbe8]
16        ttk::style map MeinStil.Vertical.TScrollbar ↩
            -arrowcolor [list active $Farbe1 disabled $Farbe8↩
            ]
17      }
18    }
19  }
20
21  set Farbe1 #0058A2 ; # dunkelblau
22  set Farbe2 #68A2D4 ; # mittelblau
23  set Farbe3 #DEE8F7 ; # hellblau
24  set Farbe4 #FFFFFF ; # weiss
25  set Farbe6 #D0D0D0 ; # hellgrau
26  set Farbe7 #A0A0A0 ; # mittelgrau
27  set Farbe8 #707070 ; # dunkelgrau
28  set Farbe10 #000000 ; # schwarz
29
30  ttk::style theme use clam
31
32  ttk::style configure MeinStil.Vertical.TScrollbar ↩
      -background $Farbe2
33  ttk::style configure MeinStil.Vertical.TScrollbar ↩
      -lightcolor $Farbe1
34  ttk::style configure MeinStil.Vertical.TScrollbar ↩
      -darkcolor $Farbe2
35  ttk::style configure MeinStil.Vertical.TScrollbar ↩
      -troughcolor $Farbe3
36  ttk::style configure MeinStil.Vertical.TScrollbar ↩
      -arrowcolor $Farbe1
37  ttk::style configure MeinStil.Vertical.TScrollbar ↩
      -arrowsize 12
38
39  ttk::style map MeinStil.Vertical.TScrollbar -background [↩
      list active $Farbe2 disabled $Farbe7]
40  ttk::style map MeinStil.Vertical.TScrollbar -bordercolor [↩
      list active $Farbe1 disabled $Farbe8]
41  ttk::style map MeinStil.Vertical.TScrollbar -lightcolor [↩
      list active $Farbe2 disabled $Farbe7]
42  ttk::style map MeinStil.Vertical.TScrollbar -darkcolor [↩
      list active $Farbe2 disabled $Farbe8]
43  ttk::style map MeinStil.Vertical.TScrollbar -arrowcolor [↩
      list active $Farbe1 disabled $Farbe8]
44
45  set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav ↩
      Heinrich Ida Julius Kaufmann}
46  set Liste [concat $Liste {Ludwig Martha Nordpol Otto Paula↩
       Quelle Richard Samuel Schule Theodor}]
47  set Liste [concat $Liste {Ulrich Viktor Wilhelm Xanthippe ↩
      Ypsilon Zacharias}]
48
49  ttk::frame .fr
```

```
50 listbox .fr.lbox -width 15 -height 15 -xscrollcommand {
      .fr.sbX set} -yscrollcommand {.fr.sbY set}
      -listvariable Liste
51 ttk::scrollbar .fr.sbX -orient horizontal -command {
      .fr.lbox xview}
52 ttk::scrollbar .fr.sbY -style MeinStil.Vertical.TScrollbar
       -command {.fr.lbox yview}
53
54 pack .fr.sbX -side bottom -fill x
55 pack .fr.sbY -side right -fill y
56 pack .fr.lbox -side top -expand yes -fill both
57 pack .fr -side top -expand yes -fill both
58
59 bind .fr.sbY <Enter> {StatusPruefen}
60
61 ttk::button .btKlein -text "Klein" -command {ttk::style
      configure MeinStil.Vertical.TScrollbar -arrowsize 10}
62 ttk::button .btGross -text "Gross" -command {ttk::style
      configure MeinStil.Vertical.TScrollbar -arrowsize 20}
63
64 pack .btKlein -side left
65 pack .btGross -side right
```

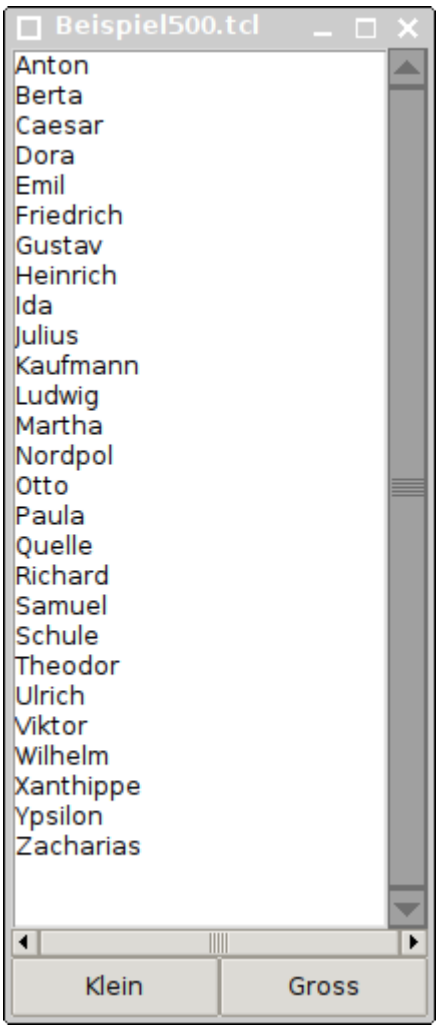
Klick auf den Button `Klein`:



Klick auf den Button `Gross`:



Wenn man das Fenster mit der Maus größer (länger) zieht, wird die Scroll-Leiste automatisch deaktiviert (Status `disabled`):



Da die Scroll-Leiste auch im Status `disabled` aktiviert wird, wenn der Mauszeiger über die Scroll-Leiste bewegt wird, wird dementsprechend auch die Farbsetzung gemäß des Status `active` angewendet (sie hat dann statt der Grautöne wieder die Blautöne). Um das zu verhindern, wird in Zeile 59 geprüft, ob der Mauszeiger in die Scroll-Leiste hineinbewegt wird. Es wird dann die Prozedur `StatusPruefen` aufgerufen, die je nach Status der Scroll-Leiste die Farben setzt.

54.3.13 Combobox

Listing 54.23: Combobox (Beispiel499.tcl)

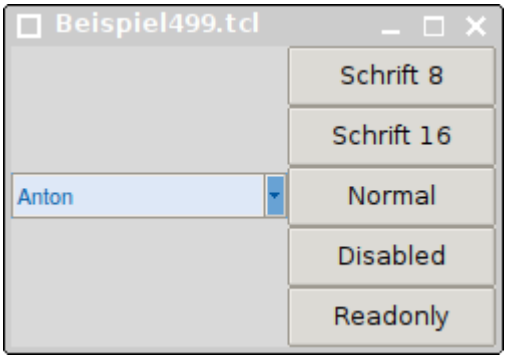
```
1 #!/usr/bin/env wish
2
3 set Farbe1 #0058A2 ; # dunkelblau
4 set Farbe2 #68A2D4 ; # mittelblau
5 set Farbe3 #DEE8F7 ; # hellblau
6 set Farbe4 #FFFFFF ; # weiss
7 set Farbe6 #D0D0D0 ; # hellgrau
8 set Farbe7 #A0A0A0 ; # mittelgrau
```

```
9  set Farbe8 #707070 ; # dunkelgrau
10 set Farbe10 #000000 ; # schwarz
11
12 font create MeineSchrift -family Helvetica -size 12 -slant
       roman -weight normal
13
14 ttk::style theme use clam
15
16 ttk::style configure MeinStil.TCombobox -background
       $Farbe2
17 ttk::style configure MeinStil.TCombobox -foreground
       $Farbe1
18 ttk::style configure MeinStil.TCombobox -fieldbackground
       $Farbe3
19 ttk::style configure MeinStil.TCombobox -arrowcolor
       $Farbe1
20 ttk::style configure MeinStil.TCombobox -arrowsize 16
21 ttk::style configure MeinStil.TCombobox -selectbackground
       $Farbe1
22 ttk::style configure MeinStil.TCombobox -selectforeground
       $Farbe3
23 ttk::style configure MeinStil.TCombobox -insertcolor
       $Farbe1
24 ttk::style configure MeinStil.TCombobox -font MeineSchrift
25
26 ttk::style map MeinStil.TCombobox -background [list
       disabled $Farbe6 readonly $Farbe2]
27 ttk::style map MeinStil.TCombobox -foreground [list
       disabled $Farbe8 readonly $Farbe1]
28 ttk::style map MeinStil.TCombobox -fieldbackground [list
       disabled $Farbe6 readonly $Farbe3]
29 ttk::style map MeinStil.TCombobox -arrowcolor [list
       disabled $Farbe8 readonly $Farbe1]
30
31 # die Scrollbars innerhalb der Combobox
32 ttk::style configure Vertical.TScrollbar -background
       $Farbe2
33 ttk::style configure Vertical.TScrollbar -lightcolor
       $Farbe1
34 ttk::style configure Vertical.TScrollbar -darkcolor
       $Farbe2
35 ttk::style configure Vertical.TScrollbar -troughcolor
       $Farbe3
36 ttk::style configure Vertical.TScrollbar -arrowcolor
       $Farbe1
37 ttk::style configure Vertical.TScrollbar -arrowsize 12
38
39 ttk::style map Vertical.TScrollbar -background [list
       active $Farbe2 disabled $Farbe4]
40
41 # fuer die aufgeklappte Comobox gilt:
42 option add *TCombobox*Listbox.font MeineSchrift
43 option add *TCombobox*Listbox.background $Farbe3
```

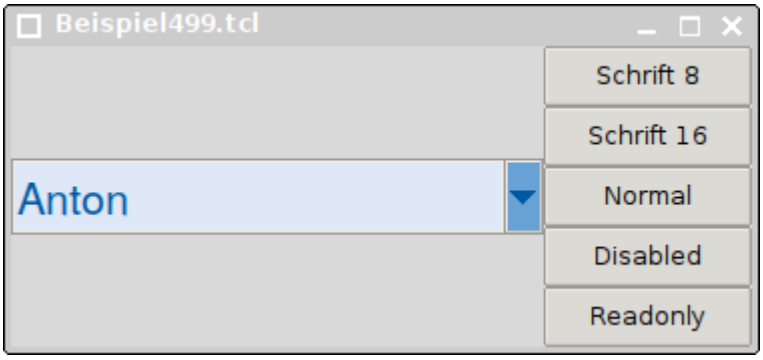
```
44 option add *TCombobox*Listbox.foreground $Farbe1
45 option add *TCombobox*Listbox.selectBackground $Farbe1
46 option add *TCombobox*Listbox.selectForeground $Farbe3
47
48 set Liste {Anton Berta Caesar Dora Emil Friedrich Gustav ↩
     Heinrich Ida Julius Kaufmann Ludwig Martha Nordpol Otto↩
      Paula Quelle Richard Samuel Theodor Ulrich Viktor ↩
     Wilhelm Xanthippe Ypsilon Zacharias}
49 set Auswahl [lindex $Liste 0]
50 ttk::combobox .cb -values $Liste -textvariable Auswahl ↩
     -font MeineSchrift -style MeinStil.TCombobox
51 pack .cb -side left
52
53 ttk::button .btKlein -text "Schrift 8" -command {
54   font configure MeineSchrift -size 8
55   ttk::style configure MeinStil.TCombobox -arrowsize 12
56   ttk::style configure Vertical.TScrollbar -arrowsize 10
57   set Auswahl $Auswahl
58 }
59 ttk::button .btGross -text "Schrift 16" -command {
60   font configure MeineSchrift -size 16
61   ttk::style configure MeinStil.TCombobox -arrowsize 20
62   ttk::style configure Vertical.TScrollbar -arrowsize 18
63   set Auswahl $Auswahl
64 }
65 pack .btKlein -side top
66 pack .btGross -side top
67
68 ttk::button .btNormal -text "Normal" -command {.cb ↩
     configure -state normal}
69 ttk::button .btDisabled -text "Disabled" -command {.cb ↩
     configure -state disabled}
70 ttk::button .btReadonly -text "Readonly" -command {.cb ↩
     configure -state readonly}
71 pack .btNormal -side top
72 pack .btDisabled -side top
73 pack .btReadonly -side top
```

Einstellung der Schrift: Zeilen 12, 24, 42, 50, 54 und 60.

Klick auf den Button `Schrift 8`:

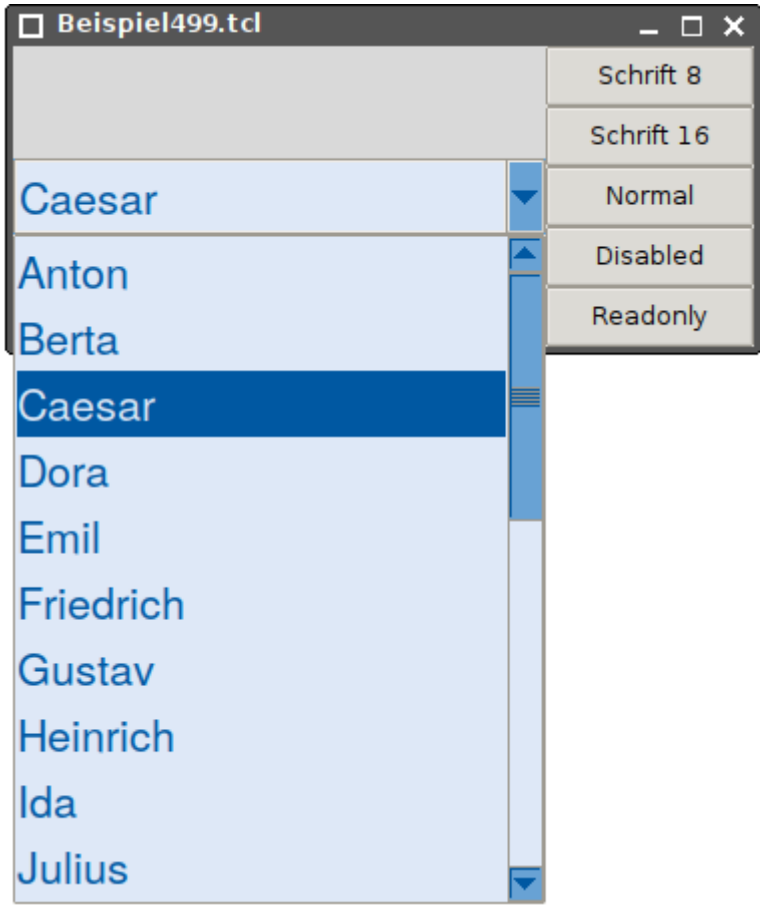


Klick auf den Button `Schrift 16`:

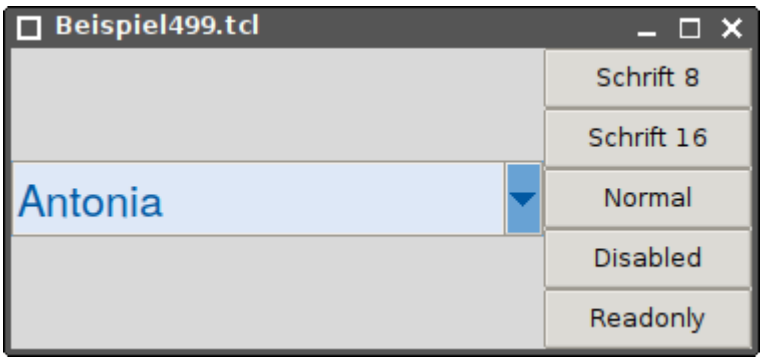


Damit der Text bei Wahl einer anderen Schriftgröße innerhalb der Combobox richtig platziert wird, muss die scheinbar sinnlose Anweisung `set Auswahl $Auswahl` ausgeführt werden (Zeilen 57 und 63).

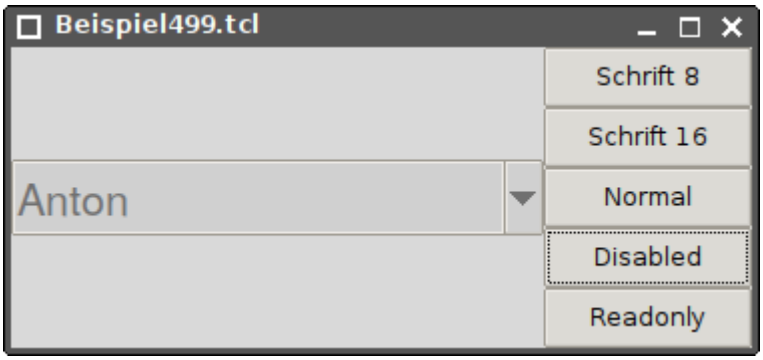
Klick auf einen Eintrag:



Eingabe eines eigenen Texts:



Klick auf den Button `Disabled`:



54.3.14 Scale

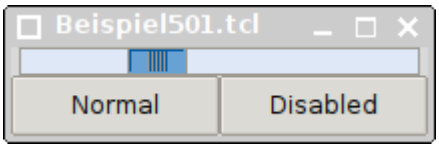
Im nachstehenden Beispiel wird das Layout des horizontalen (waagrechten) Schiebereglers verändert. Die Befehle gelten analog auch für den vertikalen Schieberegler. In diesem Fall heißt es dann `Vertical.TScale` statt `Horizontal.TScale`.

Listing 54.24: Scale (Beispiel501.tcl)

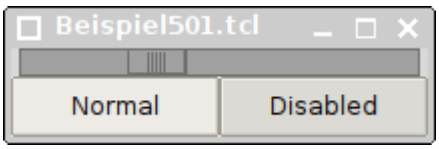
```
#!/usr/bin/env wish

proc StatusPruefen {} {
  uplevel #0 {
    if {[.sc instate disabled]} {
      ttk::style map MeinStil.Horizontal.TScale ↩
         -background [list active $Farbe7 disabled $Farbe7↩
         ]
      ttk::style map MeinStil.Horizontal.TScale ↩
         -bordercolor [list active $Farbe8 disabled ↩
         $Farbe8]
      ttk::style map MeinStil.Horizontal.TScale ↩
         -lightcolor [list active $Farbe7 disabled $Farbe7↩
         ]
      ttk::style map MeinStil.Horizontal.TScale -darkcolor↩
          [list active $Farbe8 disabled $Farbe8]
    } else {
      ttk::style map MeinStil.Horizontal.TScale ↩
         -background [list active $Farbe2 disabled $Farbe7↩
         ]
```

```
12        ttk::style map MeinStil.Horizontal.TScale ↩
             -bordercolor [list active $Farbe1 disabled ↩
             $Farbe8]
13        ttk::style map MeinStil.Horizontal.TScale ↩
             -lightcolor [list active $Farbe2 disabled $Farbe7↩
             ]
14        ttk::style map MeinStil.Horizontal.TScale -darkcolor↩
              [list active $Farbe2 disabled $Farbe8]
15      }
16    }
17  }
18
19  set Farbe1 #0058A2 ; # dunkelblau
20  set Farbe2 #68A2D4 ; # mittelblau
21  set Farbe3 #DEE8F7 ; # hellblau
22  set Farbe4 #FFFFFF ; # weiss
23  set Farbe6 #D0D0D0 ; # hellgrau
24  set Farbe7 #A0A0A0 ; # mittelgrau
25  set Farbe8 #707070 ; # dunkelgrau
26  set Farbe10 #000000 ; # schwarz
27
28  ttk::style theme use clam
29
30  ttk::style configure MeinStil.Horizontal.TScale ↩
       -background $Farbe2
31  ttk::style configure MeinStil.Horizontal.TScale ↩
       -lightcolor $Farbe1
32  ttk::style configure MeinStil.Horizontal.TScale -darkcolor↩
        $Farbe2
33  ttk::style configure MeinStil.Horizontal.TScale ↩
       -troughcolor $Farbe3
34
35  ttk::style map MeinStil.Horizontal.TScale -background [↩
       list active $Farbe2 disabled $Farbe7]
36  ttk::style map MeinStil.Horizontal.TScale -bordercolor [↩
       list active $Farbe1 disabled $Farbe8]
37  ttk::style map MeinStil.Horizontal.TScale -lightcolor [↩
       list active $Farbe2 disabled $Farbe7]
38  ttk::style map MeinStil.Horizontal.TScale -darkcolor [list↩
        active $Farbe2 disabled $Farbe8]
39
40  ttk::style configure Disabled.MeinStil.Horizontal.TScale ↩
       -troughcolor $Farbe7
41
42  ttk::scale .sc -style MeinStil.Horizontal.TScale -orient ↩
       horizontal -from 0 -to 100 -length 200 -variable Wert
43
44  pack .sc -side top
45
46  bind .sc <Enter> {StatusPruefen}
47
48  ttk::button .btNormal -text "Normal" -command {
49    .sc state !disabled ;# normal funktioniert nicht
```

```
50    .sc configure -style MeinStil.Horizontal.TScale
51  }
52  ttk::button .btDisabled -text "Disabled" -command {
53    .sc state disabled
54    .sc configure -style Disabled.MeinStil.Horizontal.TScale
55  }
56  pack .btNormal -side left
57  pack .btDisabled -side left
```



Klick auf den Button `Disabled`:



In der Zeile 40 wird ein zusätzlicher Stil erstellt, der die Darstellung des `disabled`-Zustands übernimmt. Dieser Stil wird in Zeile 55 zugeordnet.

Da der Schieberegler auch im Status `disabled` aktiviert wird, wenn der Mauszeiger darüberbewegt wird, wird dementsprechend auch die Farbsetzung gemäß des Status `active` angewendet (sie hat dann statt der Grautöne wieder die Blautöne). Um das zu verhindern, wird in Zeile 47 geprüft, ob der Mauszeiger in den Schieberegler hineinbewegt wird. Es wird dann die Prozedur `StatusPruefen` aufgerufen, die je nach Status des Schiebereglers die Farben setzt.

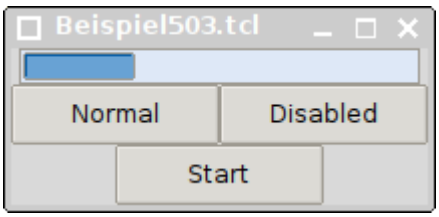
54.3.15 Progressbar

Im nachstehenden Beispiel wird das Layout des horizontalen (waagrechten) Fortschrittsbalkens verändert. Die Befehle gelten analog auch für den vertikalen Fortschrittsbalken. In diesem Fall heißt es dann `Vertical.TProgressbar` statt `Horizontal.TProgressbar`.

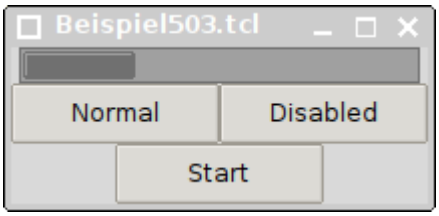
Listing 54.25: Progressbar (Beispiel503.tcl)

```
1  #!/usr/bin/env wish
2
3  proc Schleife {Element} {
4    for {set Zaehler 0} {$Zaehler <= 30} {incr Zaehler} {
5      after 30
6      $Element configure -value $Zaehler
7      update idletask
8    }
9  }
10
11 set Farbe1 #0058A2 ; # dunkelblau
12 set Farbe2 #68A2D4 ; # mittelblau
```

```
13 set Farbe3 #DEE8F7 ; # hellblau
14 set Farbe4 #FFFFFF ; # weiss
15 set Farbe6 #D0D0D0 ; # hellgrau
16 set Farbe7 #A0A0A0 ; # mittelgrau
17 set Farbe8 #707070 ; # dunkelgrau
18 set Farbe10 #000000 ; # schwarz
19
20 ttk::style theme use clam
21
22 ttk::style configure MeinStil.Horizontal.TProgressbar ↩
      -background $Farbe2
23 ttk::style configure MeinStil.Horizontal.TProgressbar ↩
      -lightcolor $Farbe1
24 ttk::style configure MeinStil.Horizontal.TProgressbar ↩
      -darkcolor $Farbe2
25 ttk::style configure MeinStil.Horizontal.TProgressbar ↩
      -troughcolor $Farbe3
26
27 ttk::style map MeinStil.Horizontal.TProgressbar ↩
      -background [list active $Farbe2 disabled $Farbe8]
28 ttk::style map MeinStil.Horizontal.TProgressbar ↩
      -bordercolor [list active $Farbe1 disabled $Farbe8]
29 ttk::style map MeinStil.Horizontal.TProgressbar ↩
      -lightcolor [list active $Farbe2 disabled $Farbe7]
30 ttk::style map MeinStil.Horizontal.TProgressbar -darkcolor↩
       [list active $Farbe2 disabled $Farbe8]
31
32 ttk::style configure ↩
      Disabled.MeinStil.Horizontal.TProgressbar -troughcolor ↩
      $Farbe7
33
34 set Wert 0
35 ttk::progressbar .pb -style ↩
      MeinStil.Horizontal.TProgressbar -orient horizontal ↩
      -maximum 100 -length 200 -value 0
36 ttk::button .bt -text "Start" -command {Schleife .pb}
37 pack .pb -side top
38 pack .bt -side bottom
39
40 ttk::button .btNormal -text "Normal" -command {
41   .pb state !disabled
42   .pb configure -style MeinStil.Horizontal.TProgressbar
43 }
44 ttk::button .btDisabled -text "Disabled" -command {
45   .pb state disabled
46   .pb configure -style ↩
        Disabled.MeinStil.Horizontal.TProgressbar
47 }
48 pack .btNormal -side left
49 pack .btDisabled -side left
```



Klick auf den Button `Disabled`:



In der Zeile 32 wird ein zusätzlicher Stil erstellt, der die Darstellung des `disabled`-Zustands übernimmt. Dieser Stil wird in Zeile 46 zugeordnet.

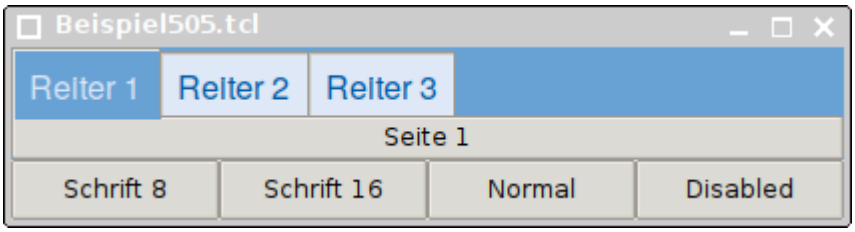
54.3.16 Notebook

Listing 54.26: Notebook (Beispiel505.tcl)

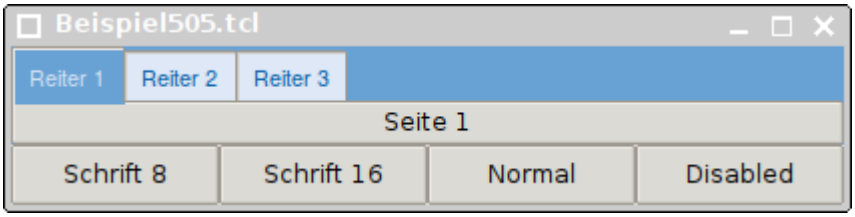
```
1  #!/usr/bin/env wish
2
3  set Farbe1 #0058A2 ; # dunkelblau
4  set Farbe2 #68A2D4 ; # mittelblau
5  set Farbe3 #DEE8F7 ; # hellblau
6  set Farbe4 #FFFFFF ; # weiss
7  set Farbe6 #D0D0D0 ; # hellgrau
8  set Farbe7 #A0A0A0 ; # mittelgrau
9  set Farbe8 #707070 ; # dunkelgrau
10 set Farbe10 #000000 ; # schwarz
11
12 font create MeineSchrift -family Helvetica -size 12 -slant ↩
       roman -weight normal
13
14 ttk::style theme use clam
15
16 ttk::style configure MeinStil.TNotebook -background ↩
       $Farbe2
17 ttk::style configure MeinStil.TNotebook -foreground ↩
       $Farbe2
18
19 ttk::style configure MeinStil.TNotebook.Tab -background ↩
       $Farbe3
20 ttk::style configure MeinStil.TNotebook.Tab -foreground ↩
       $Farbe1
21 ttk::style configure MeinStil.TNotebook.Tab -font ↩
       MeineSchrift
22
```

```
23 ttk::style map MeinStil.TNotebook.Tab -foreground [list active $Farbe2 disabled $Farbe8 selected $Farbe3]
24 ttk::style map MeinStil.TNotebook.Tab -background [list active $Farbe1 disabled $Farbe7 selected $Farbe2]
25
26 ttk::notebook .n -style MeinStil.TNotebook
27 ttk::frame .n.f1
28 ttk::frame .n.f2
29 ttk::frame .n.f3
30 .n add .n.f1 -text "Reiter 1"
31 .n add .n.f2 -text "Reiter 2"
32 .n add .n.f3 -text "Reiter 3"
33
34 ttk::label .n.f1.l1 -text "Seite 1"
35 ttk::label .n.f2.l1 -text "Seite 2"
36 ttk::label .n.f3.l1 -text "Seite 3"
37
38 pack .n -fill both -expand 1
39 pack .n.f1.l1 -side top
40 pack .n.f2.l1 -side top
41 pack .n.f3.l1 -side top
42
43 .n select .n.f1
44
45 ttk::button .btKlein -text "Schrift 8" -command {font configure MeineSchrift -size 8}
46 ttk::button .btGross -text "Schrift 16" -command {font configure MeineSchrift -size 16}
47 pack .btKlein -side left
48 pack .btGross -side left
49
50 ttk::button .btNormal -text "Normal" -command {
51   .n tab .n.f1 -state normal
52 }
53 ttk::button .btDisabled -text "Disabled" -command {
54   .n tab .n.f1 -state disabled
55 }
56 pack .btNormal -side left
57 pack .btDisabled -side left
```

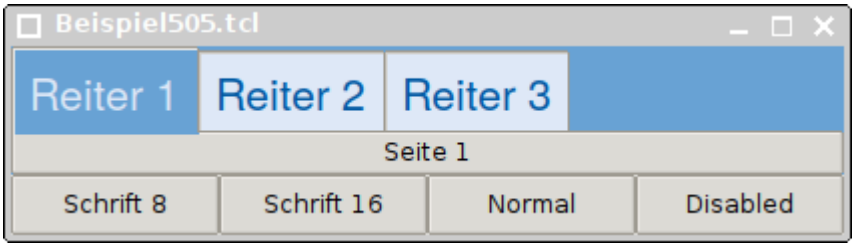
Einstellung der Schrift: Zeilen 12, 21, 45 und 46.
Nach dem Programmstart:



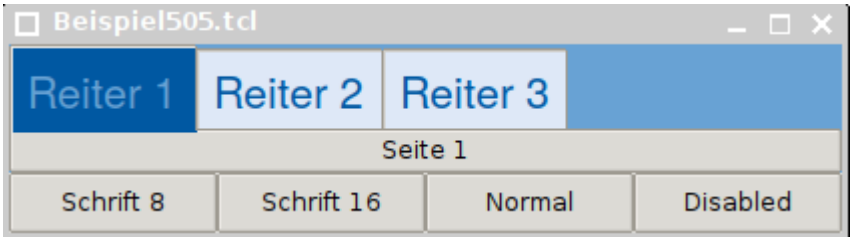
Klick auf den Button `Schrift 8`:



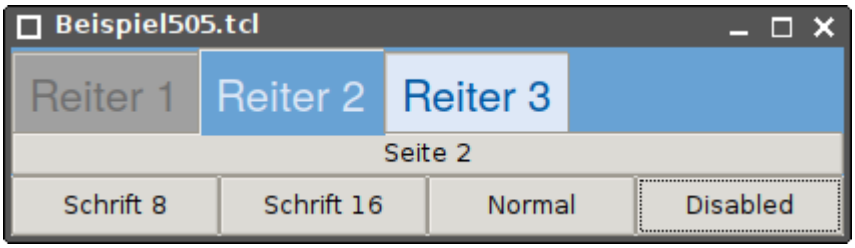
Klick auf den Button `Schrift 16`:



Maus ist über dem Reiter 1:



Klick auf den Button `Disabled`:

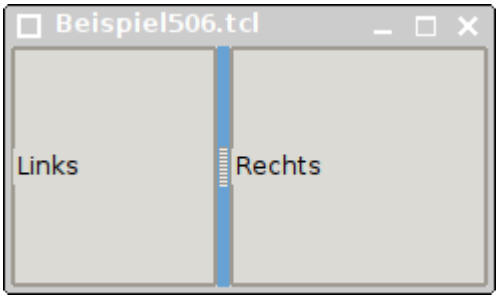


54.3.17 Panedwindow

Listing 54.27: Panedwindow (Beispiel506.tcl)

```
#!/usr/bin/env wish

set Farbe1 #0058A2 ; # dunkelblau
set Farbe2 #68A2D4 ; # mittelblau
set Farbe3 #DEE8F7 ; # hellblau
set Farbe4 #FFFFFF ; # weiss
set Farbe6 #D0D0D0 ; # hellgrau
set Farbe7 #A0A0A0 ; # mittelgrau
set Farbe8 #707070 ; # dunkelgrau
```

```
set Farbe10 #000000 ; # schwarz

ttk::style theme use clam

ttk::style configure MeinStil.TPanedwindow -background $Farbe2

ttk::panedwindow .p -style MeinStil.TPanedwindow -orient horizontal
ttk::frame .p.f1 -borderwidth 1 -relief solid
ttk::frame .p.f2 -borderwidth 1 -relief solid
.p add .p.f1
.p add .p.f2

ttk::label .p.f1.l1 -text "Links" -anchor e -justify left
ttk::label .p.f2.l1 -text "Rechts" -anchor e -justify left

pack .p -fill both -expand 1
pack .p.f1.l1 -side left
pack .p.f2.l1 -side left
```



54.3.18 Treeview

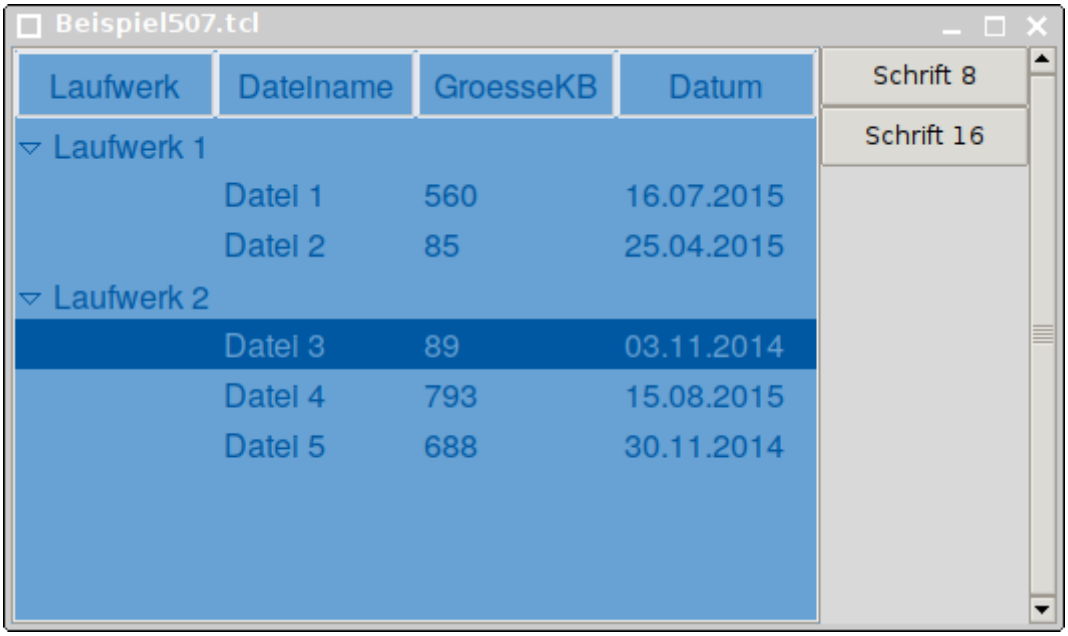
Listing 54.28: Treeview (Beispiel507.tcl)

```
#!/usr/bin/env wish

set Farbe1 #0058A2 ; # dunkelblau
set Farbe2 #68A2D4 ; # mittelblau
set Farbe3 #DEE8F7 ; # hellblau
set Farbe4 #FFFFFF ; # weiss
set Farbe6 #D0D0D0 ; # hellgrau
set Farbe7 #A0A0A0 ; # mittelgrau
set Farbe8 #707070 ; # dunkelgrau
set Farbe10 #000000 ; # schwarz

font create MeineSchrift -family Helvetica -size 12 -slant roman -weight normal

ttk::style theme use clam

ttk::style configure MeinStil.Treeview -background $Farbe2
```

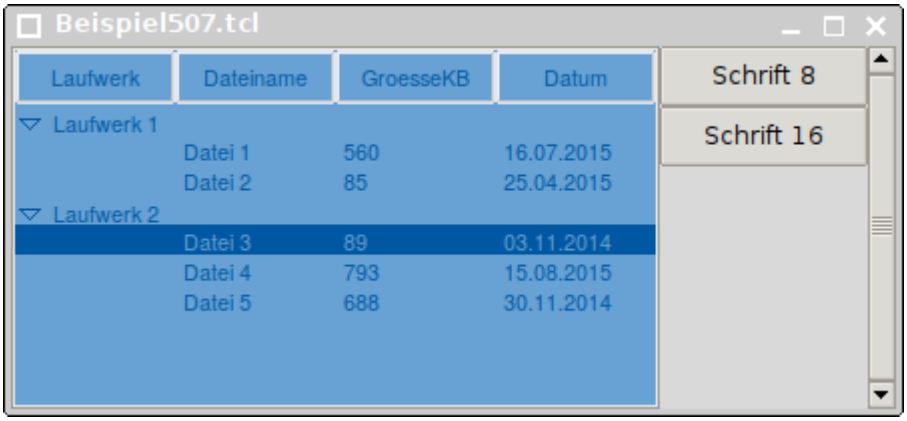
```
17 ttk::style configure MeinStil.Treeview -foreground $Farbe1
18 ttk::style configure MeinStil.Treeview -fieldbackground ↩
      $Farbe2
19 ttk::style configure MeinStil.Treeview -font MeineSchrift
20 ttk::style configure MeinStil.Treeview -rowheight 25
21
22 ttk::style map MeinStil.Treeview -background [list ↩
      selected $Farbe1]
23 ttk::style map MeinStil.Treeview -foreground [list ↩
      selected $Farbe2]
24
25 ttk::style configure MeinStil.Treeview.Heading -background↩
       $Farbe2
26 ttk::style configure MeinStil.Treeview.Heading -foreground↩
       $Farbe1
27 ttk::style configure MeinStil.Treeview.Heading ↩
      -bordercolor $Farbe3
28 ttk::style configure MeinStil.Treeview.Heading -font ↩
      MeineSchrift
29
30 ttk::style map MeinStil.Treeview.Heading -background [list↩
       active $Farbe1]
31 ttk::style map MeinStil.Treeview.Heading -foreground [list↩
       active $Farbe2]
32
33 ttk::style configure MeinStil.Treeview.Cell -padding [list↩
       0 0 0 0]
34
35 set Dateien1 {{"Datei 1" "560" "16.07.2015"} {"Datei 2" "↩
      85" "25.04.2015"}}
36 set Dateien2 {{"Datei 3" "89" "03.11.2014"} {"Datei 4" "↩
      793" "15.08.2015"} {"Datei 5" "688" "30.11.2014"}}
37
38 ttk::treeview .tv -style MeinStil.Treeview -columns {↩
      Dateiname GroesseKB Datum} -yscrollcommand {.sbY set}
39
40 .tv column #0 -width 100
41 .tv column #1 -width 100
42 .tv column #2 -width 100
43 .tv column #3 -width 100
44
45 .tv heading #0 -text "Laufwerk"
46 .tv heading Dateiname -text "Dateiname"
47 .tv heading GroesseKB -text "GroesseKB"
48 .tv heading Datum -text "Datum"
49
50 .tv insert {} end -id Laufwerk1 -text "Laufwerk 1"
51 .tv insert {} end -id Laufwerk2 -text "Laufwerk 2"
52
53 foreach Element $Dateien1 {
54   .tv insert Laufwerk1 end -values $Element
55 }
56
```

```
57 foreach Element $Dateien2 {
58   .tv insert Laufwerk2 end -values $Element
59 }
60
61 ttk::scrollbar .sbY -command {.tv yview}
62
63 pack .sbY -side right -fill y
64 pack .tv -side left -expand yes -fill both
65
66 ttk::button .btKlein -text "Schrift 8" -command {
67   font configure MeineSchrift -size 8
68   ttk::style configure MeinStil.Treeview -rowheight 15
69   .tv column #0 -width 80
70   .tv column #1 -width 80
71   .tv column #2 -width 80
72   .tv column #3 -width 80
73
74 }
75 ttk::button .btGross -text "Schrift 16" -command {
76   font configure MeineSchrift -size 16
77   ttk::style configure MeinStil.Treeview -rowheight 35
78   .tv column #0 -width 130
79   .tv column #1 -width 130
80   .tv column #2 -width 130
81   .tv column #3 -width 130
82 }
83 pack .btKlein -side top
84 pack .btGross -side top
```

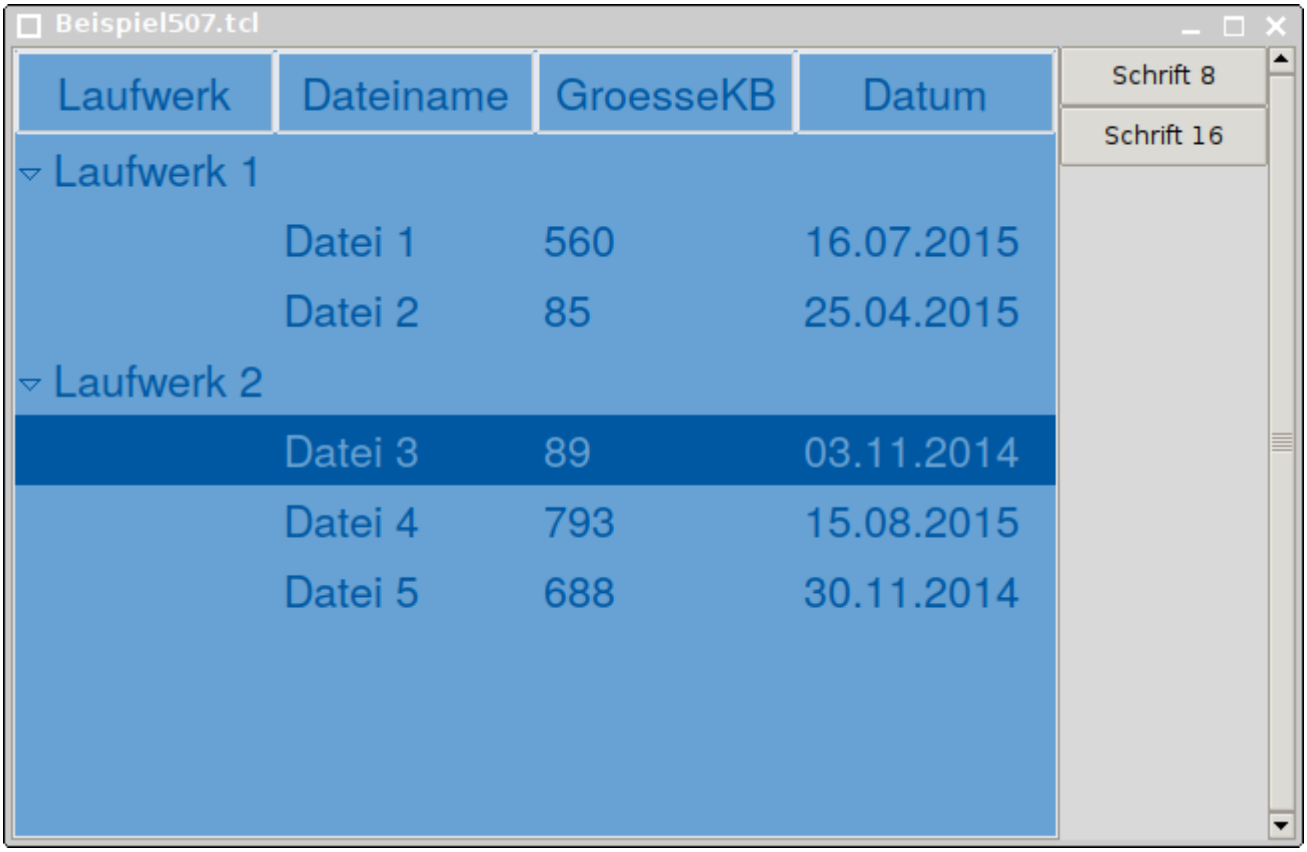
Einstellung der Schrift: Zeilen 12, 19, 28, 67 und 76. Außerdem die Zeilen 20, 68 und 77. Nach dem Programmstart:



Klick auf den Button `Schrift 8`:



Klick auf den Button `Schrift 16`:



54.3.19 Text

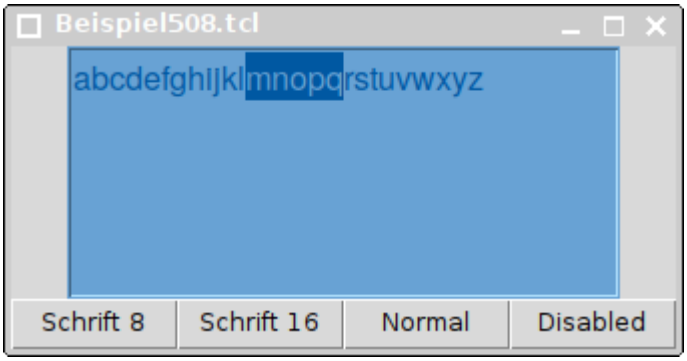
Listing 54.29: Text (Beispiel508.tcl)

```
#!/usr/bin/env wish

```

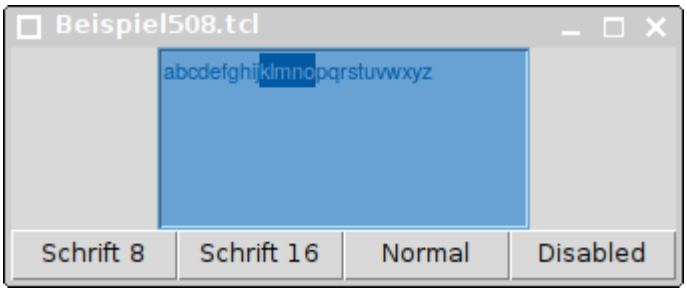
```
3  set Farbe1 #0058A2 ; # dunkelblau
4  set Farbe2 #68A2D4 ; # mittelblau
5  set Farbe3 #DEE8F7 ; # hellblau
6  set Farbe4 #FFFFFF ; # weiss
7  set Farbe6 #D0D0D0 ; # hellgrau
8  set Farbe7 #A0A0A0 ; # mittelgrau
9  set Farbe8 #707070 ; # dunkelgrau
10 set Farbe10 #000000 ; # schwarz
11
12 font create MeineSchrift -family Helvetica -size 12 -slant
       roman -weight normal
13
14 text .tx -height 5 -width 30 -wrap word -font MeineSchrift
       -foreground $Farbe1 -background $Farbe2
      -selectforeground $Farbe2 -selectbackground $Farbe1
      -highlightcolor $Farbe1 -highlightbackground $Farbe2
      -insertbackground $Farbe1 -inactiveselectbackground
      $Farbe1
15 pack .tx
16
17 ttk::button .btKlein -text "Schrift 8" -command {
18   font configure MeineSchrift -size 8
19 }
20 ttk::button .btGross -text "Schrift 16" -command {
21   font configure MeineSchrift -size 16
22 }
23 pack .btKlein -side left
24 pack .btGross -side left
25
26 ttk::button .btNormal -text "Normal" -command {
27   .tx configure -state normal
28   .tx configure -foreground $Farbe1 -background $Farbe2
        -selectforeground $Farbe2 -selectbackground $Farbe1
        -highlightcolor $Farbe1 -highlightbackground $Farbe2
        -insertbackground $Farbe1 -inactiveselectbackground
        $Farbe1
29 }
30 ttk::button .btDisabled -text "Disabled" -command {
31   .tx configure -state disabled
32   .tx configure -foreground $Farbe8 -background $Farbe7
        -selectforeground $Farbe7 -selectbackground $Farbe8
        -highlightcolor $Farbe8 -highlightbackground $Farbe7
        -insertbackground $Farbe8 -inactiveselectbackground
        $Farbe8
33 }
34 pack .btNormal -side left
35 pack .btDisabled -side left
```

Einstellung der Schrift: Zeilen 12, 14, 18 und 21.

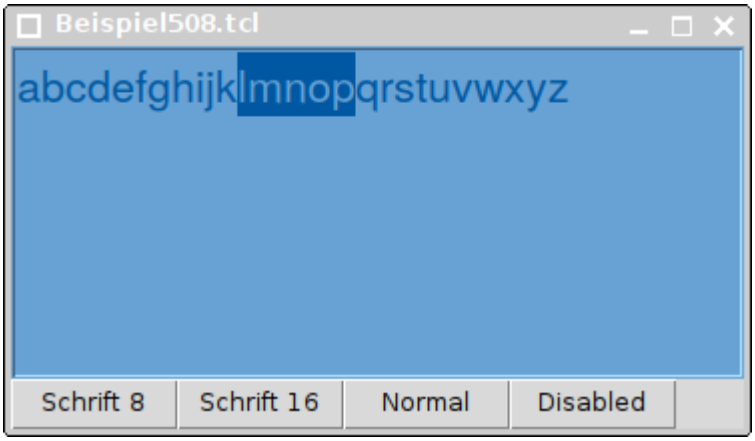
Nach dem Programmstart:



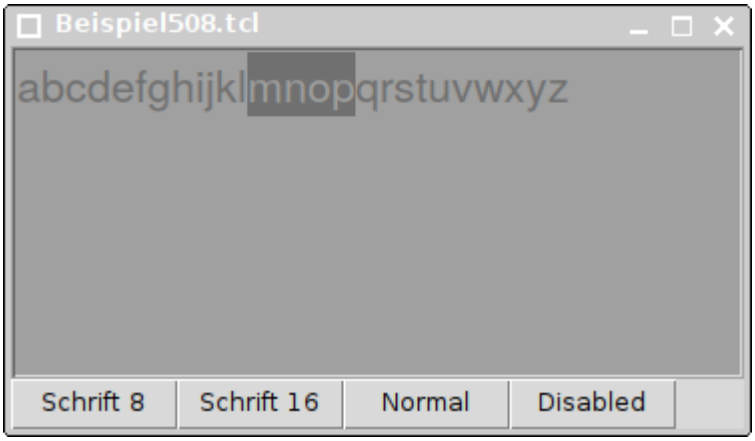
Klick auf den Button `Schrift 8`:



Klick auf den Button `Schrift 16`:



Klick auf den Button `Disabled`:


Beispiel508.tcl
abcdefghijklmnopqrstuvwxyz
Schrift 8
Schrift 16
Normal
Disabled

55 Mehrsprachige GUI

Auch die grafische Benutzeroberfläche kann mehrsprachig gestaltet werden. Wenn Sie Variablen im Text ausgeben wollen, verwenden Sie Platzhalter (siehe Kapitel 23.1 auf Seite 267 und Kapitel 12.8 auf Seite 147). Erstellen Sie zuerst eine Sprachdatei mit folgendem Inhalt:

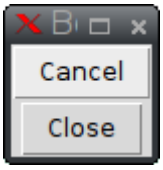
en.msg

```
1  ::msgcat::mcmset en {
2  "Abbruch" "Cancel"
3  "Schließen" "Close"
4  }
5
```

Die Datei muss unter dem Dateinamen `en.msg` in den Programmordner gespeichert werden.

Listing 55.1: Mehrsprachige GUI (Beispiel297.tcl)

```
1  #!/usr/bin/env wish
2
3  package require msgcat
4  ::msgcat::mclocale en
5  ::msgcat::mcload [file join [file dirname [info script]]]
6
7  button .btAbbruch -text [::msgcat::mc "Abbruch"] -command ↩
       exit
8  button .btSchliessen -text [::msgcat::mc "Schliessen"] ↩
       -command exit
9
10 pack .btAbbruch
11 pack .btSchliessen
```



Man erkennt, dass die im Programmcode verwendeten Texte `Abbruch` und `Schließen` durch die englischen Begriffe `Cancel` und `Close` ersetzt wurden. Die Änderung der Sprache während des laufenden Programms wirkt sich zunächst nur auf alle danach angezeigten GUI-Elemente aus. Um auch die bereits sichtbaren GUI-Elemente zu ändern, muss man die Text-Eigenschaften der Elemente im Programm neu setzen.
Erstellen Sie zuerst die deutsche Sprachdatei `de.msg` mit folgendem Inhalt:

de.msg

```
1  ::msgcat::mcmset de {
2      "Sprache" "Sprache"
3      "Info" "Info"
4      "Schliessen" "Schließen"
5      "Datei:" "Datei:"
6  }
```

Danach erstellen Sie englische Sprachdatei `en.msg` mit folgendem Inhalt:

en.msg

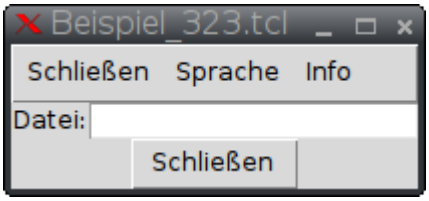
```
1  ::msgcat::mcmset en {
2      "Sprache" "Language"
3      "Info" "About"
4      "Schliessen" "Exit"
5      "Datei:" "File:"
6  }
```

Listing 55.2: Sprache während der Programmausführung ändern (Beispiel323.tcl)

```
1  #!/usr/bin/env wish
2
3  set Konf(Skriptname) [info script]
4  if {[file type $Konf(Skriptname)] == "link"} {
5    set Konf(Skriptname) [file readlink $Konf(Skriptname)]
6  }
7  ::msgcat::mclocale de ;# Alternative Sprachen: de=deutsch
     en=englisch
8  ::msgcat::mcload [file join [file dirname $Konf(Skriptname)
     )]]
9
10 proc SpracheAendern {Sprache} {
11   global Konf
12
13   switch $Sprache {
14     Deutsch {::msgcat::mclocale de}
15     English {::msgcat::mclocale en}
16   }
17   ::msgcat::mcload [file join [file dirname $Konf(
       Skriptname)]]
18
19   .lbDatei configure -text [::msgcat::mc "Datei:"]
20   .btEnde configure -text [::msgcat::mc "Schliessen"]
21
22   .m entryconfigure 0 -label [::msgcat::mc "Schliessen"]
23   .m entryconfigure 1 -label [::msgcat::mc "Sprache"]
24   .m entryconfigure 2 -label [::msgcat::mc "Info"]
25
26   update idletasks
27 }
28
```

```
29 set Sprache "Deutsch"
30 set Datei ""
31
32 option add *Menu.tearOff 0
33 . configure -menu .m
34 menu .m
35 .m add command -label [::msgcat::mc "Schliessen"] -command exit
36 .m add cascade -label [::msgcat::mc "Sprache"] -menu .m.sprache
37 .m add command -label [::msgcat::mc "Info"] -command exit
38
39 menu .m.sprache
40 .m.sprache add radiobutton -label "Deutsch" -variable Sprache -command {SpracheAendern $Sprache}
41 .m.sprache add radiobutton -label "English" -variable Sprache -command {SpracheAendern $Sprache}
42
43 ttk::label .lbDatei -text [::msgcat::mc "Datei:"]
44 ttk::entry .enDatei -textvariable Datei
45 ttk::button .btEnde -text [::msgcat::mc "Schliessen"] -command exit
46
47 grid .lbDatei -row 0 -column 0
48 grid .enDatei -row 0 -column 1
49 grid .btEnde -row 1 -column 0 -columnspan 2
```

Wenn man das Programm startet, wird es in deutscher Sprache angezeigt.



Klickt man im Menü unter `Sprache` auf den Eintrag `English`, wird das Programm in englisch angezeigt.

Die Zeilen 10 bis 27 enthalten die Prozedur `SpracheAendern`. In den Zeilen 13 bis 17 wird die Sprachdatei geladen. In den Zeilen 19 und 20 wird die Beschriftung der GUI-Elemente Label und Button geändert. In den Zeilen 22 bis 24 wird die Beschriftung des Menüs geändert. In der Zeile 26 wird die GUI aktualisiert.

56 Client-Server-Kommunikation

Im Folgenden wird ein Beispiel für eine einfache Client-Server-Kommunikation gezeigt. Das Beispiel stammt von `https://www.tcl.tk/about/netserver.html`.

Listing 56.1: Server (Beispiel513Server.tcl)

```
#!/usr/bin/env tclsh

# Quelle: https://www.tcl.tk/about/netserver.html

# Echo_Server --
#       Open the server listening socket
#       and enter the Tcl event loop
#
# Arguments:
#       port    The server's port number

proc Echo_Server {port} {
  set s [socket -server EchoAccept $port]
  vwait forever
}

# Echo_Accept --
#       Accept a connection from a new client.
#       This is called after a new socket connection
#       has been created by Tcl.
#
# Arguments:
#       sock    The new socket connection to the client
#       addr    The client's IP address
#       port    The client's port number

proc EchoAccept {sock addr port} {
  global echo

  # Record the client's information

  puts "Accept $sock from $addr port $port"
  set echo(addr,$sock) [list $addr $port]

  # Ensure that each "puts" by the server
  # results in a network transmission

  fconfigure $sock -buffering line

  # Set up a callback for when the client sends data

  fileevent $sock readable [list Echo $sock]
```

```
43 }
44
45 # Echo --
46 #        This procedure is called when the server
47 #        can read data from the client
48 #
49 # Arguments:
50 #        sock     The socket connection to the client
51
52 proc Echo {sock} {
53   global echo
54
55   # Check end of file or abnormal connection drop,
56   # then echo data back to the client.
57
58   if {[eof $sock] || [catch {gets $sock line}]} {
59     close $sock
60     puts "Close $echo(addr,$sock)"
61     unset echo(addr,$sock)
62   } else {
63     puts "Der Server hat vom Client empfangen: $line"
64     puts $sock "Hallo Client!"
65   }
66 }
67
68 Echo_Server 2540 ; # Port 2540
```

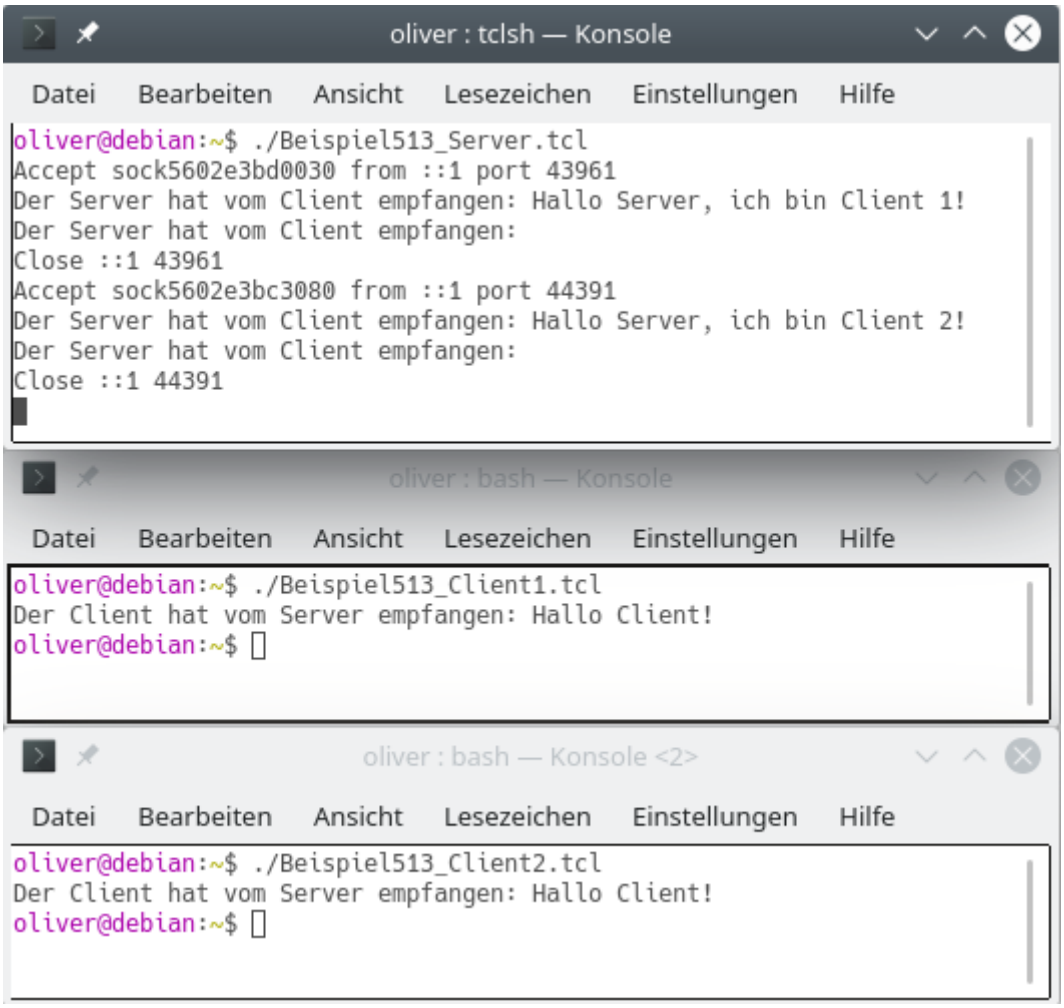
Listing 56.2: Client 1 (Beispiel513Client1.tcl)

```
1 #!/usr/bin/env tclsh
2
3 # Quelle: https://www.tcl.tk/about/netserver.html
4
5 # A client of the echo service.
6
7 proc Echo_Client {host port} {
8   set s [socket $host $port]
9   fconfigure $s -buffering line
10  return $s
11 }
12
13 # A sample client session looks like this
14 set s [Echo_Client localhost 2540]
15 puts $s "Hallo Server, ich bin Client 1!"
16 gets $s line
17 puts "Der Client hat vom Server empfangen: $line"
```

Listing 56.3: Client 2 (Beispiel513Client2.tcl)

```
1 #!/usr/bin/env tclsh
2
3 # Quelle: https://www.tcl.tk/about/netserver.html
4
```

```
5  # A client of the echo service.
6
7  proc Echo_Client {host port} {
8    set s [socket $host $port]
9    fconfigure $s -buffering line
10   return $s
11 }
12
13 # A sample client session looks like this
14 set s [Echo_Client localhost 2540]
15 puts $s "Hallo Server, ich bin Client 2!"
16 gets $s line
17 puts "Der Client hat vom Server empfangen: $line"
```



57 Programm weitergeben

57.1 Weitergabe des Quellcodes

Wenn Sie Ihr Programm in Quellcode-Form an einen Dritten weitergeben, muss der Empfänger ebenfalls Tcl/Tk auf seinem PC installiert haben. Das Programm startet der Anwender genauso wie Sie. Details finden Sie im Kapital `Tcl/Tk-Programm starten`.

57.2 Weitergabe als Binärdatei

Um es dem Empfänger bequem zu machen, können Sie Ihr Programm zusammen mit Tcl/Tk in eine einzige, ausführbare Datei packen. Der Empfänger braucht dann Tcl/Tk nicht installieren und kann direkt Ihre Datei starten. Dazu gibt es die beiden Programme

- `tclexecomp` (`http://tclexecomp.sourceforge.net/`)
- `freewrap` (`http://freewrap.sourceforge.net/`)

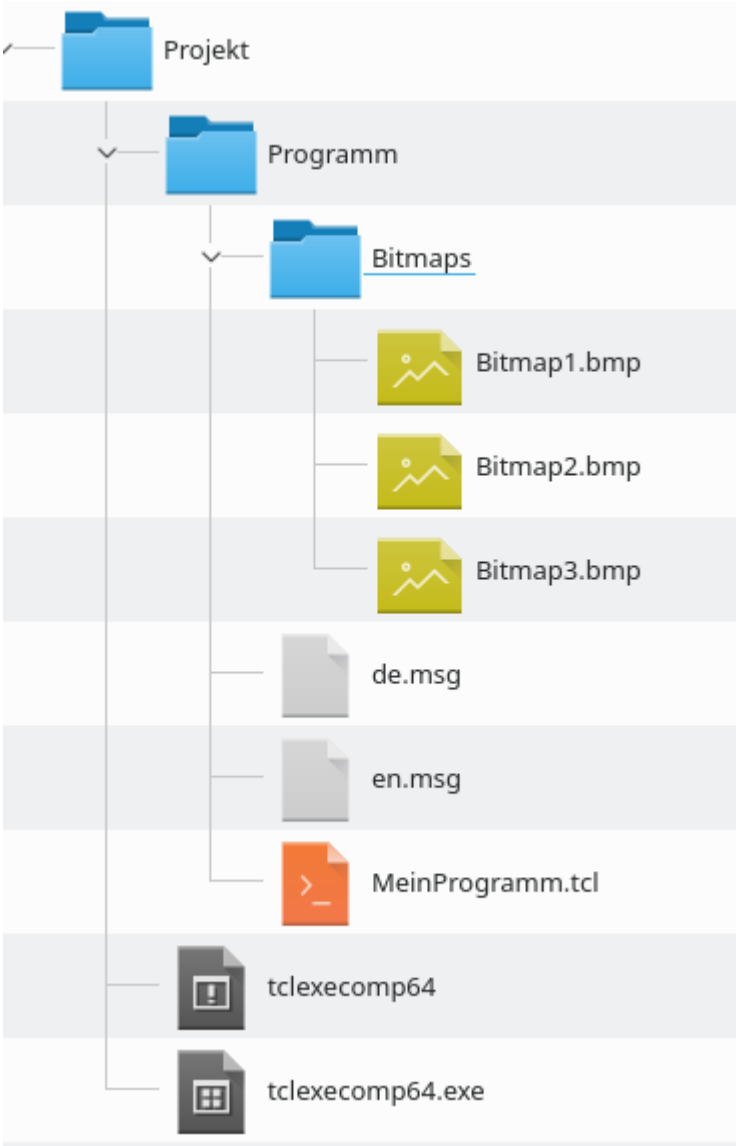
Das Programm `tclexecomp` basiert auf `freewrap`, enthält aber mehr Module. Im Folgenden wird gezeigt, wie Sie Ihr Programm mit `tclexecomp` zu einer ausführbaren Datei zusammenstellen.

Im folgenden Beispiel wird gezeigt, wie Sie unter Linux eine ausführbare Windowsdatei erzeugen.

Erstellen Sie einen Ordner `Projekt`.

Kopieren Sie in diesen Ordner die beiden Dateien `tclexecomp64` und `tclexecomp64.exe`.

Erstellen Sie innerhalb des Ordners `Projekt` den Ordner `Programm`. In diesen Ordner kopieren Sie alle Programmdateien (inklusive der zugehörigen Unterordner). In diesem Beispiel sieht das Projekt wie folgt aus:



Starten Sie eine Konsole und navigieren Sie in den Ordner `Projekt`.

Machen Sie die Datei `tclexecomp64` ausführbar (z. B. mit `chmod a+x tclexecomp64`).

Geben Sie dann folgenden Befehl ein:

```
./tclexecomp64 Programm/MeinProgramm.tcl -w tclexecomp64.exe
-o Programmname.exe Programm Programm/* Programm/Bitmaps/* -forcewrap
```

```
Projekt : bash — Konsole
Datei  Bearbeiten  Ansicht  Lesezeichen  Einstellungen  Hilfe
oliver@debian:~$ cd Projekt/
oliver@debian:~/Projekt$ chmod a+x tclexecomp64
oliver@debian:~/Projekt$ ./tclexecomp64 Programm/MeinProgramm.tcl -w tclexecomp64.exe
-o MeinProgramm.exe Programm/ Programm/* Programm/Bitmaps/* -forcewrap
```

Die einzelnen Parameter des Befehls sind:

Tabelle 57.1: Erstellung einer Windows-Datei unter Linux

Parameter	Beschreibung
`./tclexecomp64`	Startet `tclexecomp`
`Programm/MeinProgramm.tcl`	Das Tcl-Programm (im Ordner `Programm`)
`-w tclexecomp64.exe`	Die Tcl/Tk-Version, die in die ausführbare Datei integriert werden soll (`tclexecomp64.exe`, wenn Sie eine ausführbare Windows-Datei erstellen wollen oder `tclexecomp64` für eine ausführbare Linux-Datei)
`-o Programmname.exe`	Name der ausführbaren Datei
`Programm`	Der Ordner `Programm` wird hinzugefügt
`Programm/*`	Alle Dateien und Ordner innerhalb des Ordners `Programm` werden hinzugefügt. Dadurch wird auch der Ordner `Bitmaps` hinzugefügt, jedoch ohne die darin enthaltenen Dateien.
`Programm/Bitmaps/*`	Alle Dateien und Ordner innerhalb des Ordners `Programm/Bitmaps` werden hinzugefügt

Wie man an dem Befehl sieht, werden nicht automatisch alle Unterordner hinzugefügt. Stattdessen müssen die Unterordner einzeln benannt werden.
Im Ergebnis wird die Datei `Programmname.exe` erzeugt, die Sie an den Empfänger weitergeben können. Es handelt sich um eine unter Windows ausführbare Datei.
Wenn Sie unter Linux eine ausführbare Linux-Datei erstellen möchten, brauchen Sie in dem Befehl nur die Parameter `-w` und `-o` wie folgt ändern:

Tabelle 57.2: Erstellung einer Linux-Datei unter Linux

Parameter	Beschreibung
`./tclexecomp64`	Startet `tclexecomp`
`Programm/MeinProgramm.tcl`	Das Tcl-Programm
`-w tclexecomp64`	Die Tcl/Tk-Version, die in die ausführbare Datei integriert werden soll
`-o Programmname`	Name der ausführbaren Datei
`Programm`	Der Ordner `Programm` wird hinzugefügt
`Programm/*`	Alle Dateien und Ordner innerhalb des Ordners `Programm` werden hinzugefügt. Dadurch wird auch der Ordner `Bitmaps` hinzugefügt, jedoch ohne die darin enthaltenen Dateien.
`Programm/Bitmaps/*`	Alle Dateien und Ordner innerhalb des Ordners `Programm/Bitmaps` werden hinzugefügt

Wenn Sie unter Windows eine ausführbare Datei Ihres Projekts erstellen möchten, ist das Vorgehen grundsätzlich gleich. Um eine ausführbare Windows-Datei zu erstellen verwenden Sie folgende Parameter:

Tabelle 57.3: Erstellung einer Windows-Datei unter Windows

Parameter	Beschreibung
`./tclexecomp64.exe`	Startet `tclexecomp`
`Programm/MeinProgramm.tcl`	Das Tcl-Programm
`-w tclexecomp64.exe`	Die Tcl/Tk-Version, die in die ausführbare Datei integriert werden soll
`-o Programmname.exe`	Name der ausführbaren Datei
`Programm`	Der Ordner `Programm` wird hinzugefügt

Tabelle 57.3: Erstellung einer Windows-Datei unter Windows

Parameter	Beschreibung
`Programm/*`	Alle Dateien und Ordner innerhalb des Ordners `Programm` werden hinzugefügt. Dadurch wird auch der Ordner `Bitmaps` hinzugefügt, jedoch ohne die darin enthaltenen Dateien.
`Programm/Bitmaps/*`	Alle Dateien und Ordner innerhalb des Ordners `Programm/Bitmaps` werden hinzugefügt

Und um eine ausführbare Linux-Datei zu erstellen, legen Sie die Parameter wie folgt fest:

Tabelle 57.4: Erstellung einer Linux-Datei unter Windows

Parameter	Beschreibung
`./tclexecomp64.exe`	Startet `tclexecomp`
`Programm/MeinProgramm.tcl`	Das Tcl-Programm
`-w tclexecomp64`	Die Tcl/Tk-Version, die in die ausführbare Datei integriert werden soll
`-o Programmname`	Name der ausführbaren Datei
`Programm`	Der Ordner `Programm` wird hinzugefügt
`Programm/*`	Alle Dateien und Ordner innerhalb des Ordners `Programm` werden hinzugefügt. Dadurch wird auch der Ordner `Bitmaps` hinzugefügt, jedoch ohne die darin enthaltenen Dateien.
`Programm/Bitmaps/*`	Alle Dateien und Ordner innerhalb des Ordners `Programm/Bitmaps` werden hinzugefügt

58 Änderungshistorie

Tabelle 58.1: Änderungshistorie

Datum	Änderung
2014-10-05	Erstveröffentlichung
2014-10-26	Korrektur im Kapitel "121. Mehrsprachige GUI"
2014-12-21	Neue Kapitel: Kapitel "54. Plattformübergreifende Programme" Kapitel "106. Canvas (Zeichenfläche)" Kapitel "107. Diagramme mit Canvas und Gnuplot" Kapitel "108. Animationen erstellen" Ergänzungen in folgenden Kapiteln: Kapitel "21. Rechnen" Kapitel "25. Datum und Uhrzeit" Kapitel "32. Lokale und globale Variablen" Kapitel "45. Arrays" Kapitel "58. Datenbank sqlite3" Kapitel "99. Combobox" Kapitel "105. Menü" Kapitel "123. Layout"
2015-02-10	Ergänzungen in folgenden Kapiteln: Kapitel "9. Tcl/Tk-Programme unter Windows ausführen" Kapitel "10. Tcl/Tk-Programme unter Linux ausführen" Kapitel "37. Listen" Kapitel "40. Strings" Kapitel "85. Frame" Kapitel "106. Canvas (Zeichenfläche)"

Tabelle 58.1: Änderungshistorie

Datum	Änderung
2015-03-05	Neue Kapitel: Kapitel "17. Programm mit Argumenten (Parametern) starten" Kapitel "18. Dezimale, hexadezimale, binäre und ASCII-Darstellung von Zahlen" Kapitel "54. Umgang mit Binär-Dateien" Ergänzungen in folgenden Kapiteln: Kapitel "108. Menü" Kapitel "128. Mehrsprachige GUI"
2015-05-14	Neue Kapitel: Kapitel "50. Unterschied zwischen Array und Dictionary" Kapitel "110. TkTable" Ergänzungen in folgenden Kapiteln: Kapitel "28. Rechnen mit Datum und Uhrzeit" Korrekturen: Kapital "39. Listen", Befehl lreplace.
2015-07-05	Neue Kapitel: Kapitel "33. Prozeduren in separate Datei auslagern" Kapitel "64. Objektorientierte Programmierung" Ergänzung in folgendem Kapitel: Kapitel "69. Der Geometrie-Manager pack"
2015-08-28	Neue Kapitel: Kapitel "61. Andere Programme starten" Kapitel "112. Text" Ergänzungen in folgenden Kapitel: Kapitel "27. Datum und Uhrzeit" Kapitel "111. Treeview"
2015-09-07	Ergänzungen in folgenden Kapitel: Kapitel "61. Andere Programme starten" Kapitel "97. Bilder verwenden"

Tabelle 58.1: Änderungshistorie

Datum	Änderung
2015-09-26	Neue Kapitel: Kapitel "43. Listen expandieren mit `{*}`" Kapitel "55. Ordner durchsuchen mit fileutil::find" Kapitel "58. Konfigurationsdatei speichern und einlesen" Kapitel "66. Befehle aus einer Textdatei ausführen" Ergänzungen in folgenden Kapitel: Kapitel "12. Ausgabe von Text und Zahlen" Kapitel "49. Arrays" Kapitel "50. Ein Array an eine Prozedur übergeben" Kapitel "54. Ordner durchsuchen mit glob" Kapitel "64. Andere Programme starten" Kapitel "65. Befehle dynamisch erzeugen" Kapitel "95. Allgemeine Optionen der Elemente"
2015-11-15	Neue Kapitel: Kapitel "16. Variablenname mit Hilfe einer anderen Variablen bilden" Kapitel "21. Zahlen mit führender Null" Kapitel "37. Programmablauf im Detail (Regeln des Tcl-Interpreters)" Kapitel "43. Elemente aus einer Liste extrahieren mit foreach" Kapitel "44. Alle Kombinationen aus den Elementen einer Liste erzeugen (Permutation)" Kapitel "52. Zahl als String oder als Zahl behandeln" Ergänzungen in folgenden Kapitel: Kapitel "41. Listen" Kapitel "63. Umgang mit Binär-Dateien"

Tabelle 58.1: Änderungshistorie

Datum	Änderung
2016-03-13	Neue Kapitel: Kapitel "124. Popup-Menü (Kontextmenü)" Kapitel "125. Optionen-Menü" Kapitel "128. Canvas als Hilfsmittel, viele Elemente in einem Fenster darzustellen" Ergänzungen in folgenden Kapiteln: Kapitel "23. Zufallszahlen" Kapitel "41. Listen" Kapitel "59. Ordner durchsuchen mit fileutil::find" Kapitel "74. Datenbank sqlite3" Kapitel "80. Der Geometrie-Manager pack" Kapitel "108 Button" Kapitel "109. Entry" Kapitel "120. Panedwindow" Korrektur in folgendem Kapitel: Kapitel "135. Maus- und Tastaturereignisse (Bindings)"
2016-06-15	Neue Kapitel: Kapitel "36. Upvar und Uplevel in Prozeduren" Kapitel "102. Variablen und Befehle der Elemente sind global" Kapitel "130. Zeichenflächen ,die größer als das Fenster sind" Ergänzungen in folgenden Kapiteln: Kapitel "37. Programmablauf (Kurzform)" Kapitel "55. Ein Array an eine Prozedur übergeben" Kapitel "111. Entry" Kapitel "117. Listbox" Kapitel "124. Text" Kapitel "139. Maus- und Tastaturereignisse (Bindings)"
2016-08-10	Neue Kapitel: Kapitel "54. Mustererkennung in Strings bzw. Text ersetzen in Strings" Ergänzungen in folgenden Kapiteln: Kapitel "122. Notebook" Kapitel "131. Zeichenflächen ,die größer als das Fenster sind"

Tabelle 58.1: Änderungshistorie

Datum	Änderung
2016-09-14	Ergänzungen in folgenden Kapiteln: Kapitel "5. Hilfreiche Internetseiten zu Tcl/Tk" Kapitel "48. Strings (Text)" Kapitel "90. Elemente ausblenden" Kapitel "91. Elemente einblenden" Kapitel "119. Combobox"
2016-10-24	Neue Kapitel: Kapitel "143. Virtuelle Ereignisse" Ergänzungen in folgenden Kapiteln: Kapitel "118. Listbox"
2016-12-27	Ergänzungen in folgenden Kapiteln: Kapitel "42. Listen" Kapitel "64. Konfigurationsdatei speichern und einlesen" Kapitel "100. `tk_getOpenFile`" Kapitel "101. `tk_getSaveFile`" Kapitel "114. Labelframe" Kapitel "138. Eigenschaft eines Elements nachträglich ändern" Kapitel "150. Layout"
2017-02-04	Neue Kapitel: Kapitel "65. csv-Dateien öffnen bzw. erstellen" Kapitel "67. Matrix"
2017-06-04	Neue Kapitel: Kapitel "17. Fehlermeldungen verstehen" Ergänzungen in folgenden Kapiteln: Kapitel "21. Rechnen" Kapitel "60. Dateien und Ordner" Kapitel "109. Label"
2017-08-16	Neue Kapitel: Kapitel "82. Nur für Windows: gleichzeitige Nutzung von GUI und Konsole" Ergänzungen in folgenden Kapiteln: Kapitel "128. Treeview" Kapitel "134. Canvas (Zeichenfläche)"

Tabelle 58.1: Änderungshistorie

Datum	Änderung
2017-11-11	Ergänzungen in folgenden Kapiteln: Kapitel "128. Treeview" Kapitel "154. Layout"
2018-03-27	Neues Layout Ergänzungen in folgenden Kapiteln: Kapitel "24. Datenbank sqlite3" Kapitel "38.4. Entry" Kapitel "38.17. Text" Kapitel "46. TkTable"
2018-06-17	Ergänzungen in folgenden Kapiteln: Kapitel "41.1. Binding für ein Element" Kapitel "46. TkTable" Kapitel "48. Gnuplot"
2018-08-04	Ergänzung in folgendem Kapitel: Kapitel "6.1.1. Rechnen mit Zahlen"
2018-08-18	Ergänzung in folgendem Kapitel: Kapitel "8.5. Uplevel"
2018-08-23	Ergänzung in folgendem Kapitel: Kapitel "47.1. Zeichenfläche"

Tabelle 58.1: Änderungshistorie

Datum	Änderung
2019-02-03	Neue Kapitel: Kapitel "10.4. Vergleich von Listen mit ::struct::set" Kapitel "28.2. Fenstergröße abfragen" Kapitel "34.9. Menubutton" Kapitel "35.6. Prüfen, ob ein Element existiert" Kapitel "35.7. Größe und Position eines Elements abfragen" Kapitel "35.8. Element löschen" Kapitel "43.6. Farbe eines Bildpunkts ermitteln" Kapitel "46.3. Farbe und Schrift der Elemente festlegen" Ergänzungen in folgenden Kapiteln oder Beispielen: Kapitel "6.2.1. Formatierung, Format `%j`" Kapitel "14.1. Dateien und Ordner, Befehl `[info nameofexecutable]`" Kapitel "14.1. Dateien und Ordner, Beispiel 135" Kapitel "25. Objektorientierte Programmierung, Beispiel 342 und 477" Kapitel "31.1. Scroll-Leisten mit pack, Beispiel 463" Kapitel "34.2. Label, Option `-anchor`" Kapitel "34.7. Checkbutton, Option `-command`, Beispiel 464" Kapitel "34.8. Radiobutton, Option `-command`, Beispiel 465" Kapitel "34.11. Listbox, Beispiele 478-480" Kapitel "35.9. Element deaktivieren und aktivieren, Option `-state readonly`" Kapitel "41.1. Toplevel-Dialog, Beispiel 476"

Tabelle 58.1: Änderungshistorie

Datum	Änderung
2019-12-15	Neue Kapitel: Kapitel "3. Beispiel-Programm" Kapitel "6.3.4. Prüfen, ob eine Variable existiert" Kapitel "15. Stack (Stapel)" Kapitel "44.3. Tabelle sortieren" Kapitel "50. Client-Server-Kommunikation" Ergänzungen in folgenden Kapiteln oder Beispielen: Kapitel "11.1. Listen, Beispiele 515 und 516" Kapitel "13.2. Array an Prozedur übergeben" Kapitel "15.2. Ordner durchsuchen mit glob", Beispiele 139 und 438 Kapitel "16. Matrix, Beispiele 433 und 434" Kapitel "16.1. Matrix mittels Liste sortieren" Kapitel "36.17. Treeview, Beispiele 263 und 514" Kapitel "44. TkTable, Beispiel 133" Kapitel "45.5. Objekt bewegen, Beispiel 512"
2020-10-12	Neue Kapitel: Kapitel "16.1 Matrix kopieren" Kapitel "37.1. Einheitliche Hintergrundfarbe des Dialogs und der Elemente" Kapitel "45.7 Bildschirmfoto erstellen" Ergänzungen in folgenden Kapiteln oder Beispielen: Kapitel "7.1.1. Rechnen mit Zahlen", Funktionen `asin`, `acos`, `atan`, Beispiel 532 Kapitel "12.1. Text, Beispiele 522 bis 524" Kapitel "14.1. Dictionary", Beispiel 531 Kapitel "36.17. Treeview", Beispiele 447, 529, 530, 533, 534, 535 und 538 Kapitel "39.1. Binding für ein Element", `<KP_Enter>` Kapitel "40. Mauszeiger, Beispiele 525 und 526"

Tabelle 58.1: Änderungshistorie

Datum	Änderung
2022-01-02	Neue Kapitel: Kapitel "11.2 Duplikate entfernen ohne die Reihenfolge zu ändern" Kapitel "16. Pool" Kapitel "17. Tree (Baum)" Kapitel "18. Graph (Gerichteter Graph)" Kapitel "28. Formel-Eingabe durch den Anwender" Kapitel "26. Tcl mit C/C++ kombinieren" Kapitel "41.19 Kalender" Kapitel "41.18.1. Text-Element durchsuchen" Kapitel "46. Tooltip anzeigen" Ergänzungen in folgenden Kapiteln oder Beispielen: Kapitel "7.1.1. Rechnen mit Zahlen" Kapitel "11. Listen, Beispiele 556 und 557" Kapitel "12.1. Text, Beispiel 561" Kapitel "57. Programm weitergeben"
2022-07-26	Wegen geschäftspolitischer Änderungen seitens ActiveState wurden die Verweise auf ActiveTcl entfernt.
2023-12-02	Neue Kapitel: Kapitel "9.6 Trace" Kapitel "44.6 Mausklicks eine Zeit lang ignorieren" Kapitel "51.5 Mit der Maus zeichnen"