Hallo Ihr da draußen, hier kommt der fünfundzwanzigste Teil von „Icon auf dem Raspberry Pi“. Hoffentlich konntet Ihr den letzten Teil genießen und allen Schritten erfolgreich folgen. Wenn ja, dann seid Ihr in der Lage, Anwendungen nach Entwurf der graphischen Benutzeroberfläche mit sinnvollem Code (in Form von Callback-Routinen) zu vervollständigen. Korrekterweise sollte es mit *der einzigen* graphischen Benutzeroberfläche heißen. Denn bis jetzt wissen wir nur, wie Anwendungen entworfen und programmiert werden, auf denen sich auf dem *einzigen* Fenster etwas tut. Wir haben noch keine Vorstellung darüber, wie Anwendungen mit mehreren gleichberechtigten Fenstern miteinander kommunizieren oder wie die Ereignisschleife(n) auf Ereignisse mehrerer Fenster reagieren könnte(n).

Lösung der Übungsaufgabe:

Na ja, so richtig gab es keine Übungsaufgabe. Nur die Unentwegten sollten sich angesprochen fühlen. Wir wollten die GPIO-Kommandozentrale optisch ein wenig aufpeppen.

- Die 5V-Buttons sollen mit einem roten Hintergrund erscheinen
- Die 3V3-Buttons sollen mit einem hellroten Hintergrund erscheinen
- Die GPIO-Buttons sollen mit einem grünen Hintergrund erscheinen
- Die GND-Buttons sollen mit einem grauen Hintergrund erscheinen
- Alles, was mit der 7Segment-Anzeige zu tun hat, soll mit einem gelben Hintergrund erscheinen

Dann wollen wir den neuen Modellen A+ und B+ Tribut zollen, indem wir unsere GPIO-Kommandozentrale von 26 auf 40 Pins der GPIO-Schnittstelle erweitern. Dies bedingt dann auch, dass wir die GPIO-Library auf die Gegebenheiten dieser neuen Modelle erweitern. Und mit der Vorstellung der neuen GPIO-Library beginnen wir dann auch.

Durch Auswerten der Zeile `Revision` der Datei `/proc/cpuinfo` erhalten wir eine Informatrion über das Raspberry Pi-Modell und damit der Zuordnung / Kennzeichnung der GPIO-Pins. Diese Zeile sieht z.B. so aus:

```
Revision        : 0003
```

oder bei „überspannten“ Modellen z.B. so aus:

```
Revision        : 10000003
```

Wir müssen vier Fälle unterscheiden:

1. Revisionen 0002 und 0003 besitzen die PCB Revision 1.0
2. Revisionen 0004 bis 000f besitzen die PCB-Revision 2.0
3. Revisionen 0010 (Modell B+) und Revision 0012 (Modell A+) besitzen zusätzlich zu den 26 GPIO-Pins der PCB-Revision 2.0 noch 14 weitere Pins, die ergänzend aufzunehmen sind – allerdings sind die GPIO-Pins von P5 wieder abzuziehen (also kein GPIO28, GPIO29, GPIO30, GPIO31)
4. Revision 0011 gilt nur für das Compute Module – dieses besutzt die GPIO-Pins GPIO0 bis GPIO45.

Die nun überarbeitete Version der GPIO-Library passt sich auch weiterhin der PCB-Version des betreffenden Modells an.

Wie können wir erreichen, dass der Quellcode der GPIO-Kommandozentrale das jeweils gültige GPIO-Layout verwendet?

Die beste Möglichkeit besteht in der Anwendung von bedingtem Code. Wir erinnern uns, es ist das, was zwischen `$ifdef name` und `$endif` steht. Nur, wie bekommen wir die GPIO-Kommandozentrale dazu, `name` für die verschiedenen GPIO-Layouts zu erhalten? Denn es versteht sich von selbst, dass wir weder an Quellcodes der GPIO-Library noch der GPIO-Kommandozentrale Änderungen vornehmen möchten, nur weil wir die Anwendung auf einem anderen RaspberryPi-Modell einsetzen wollen.

Die einfachste Möglichkeit besteht darin, dass wir die Anweisungen, die die Datei `/proc/cpuinfo` auswerten, in ein eigenständiges Programm `GPIO_BOARDdetection.icn` einbauen. Dieses Tool wertet die Zeile `Revision` der Datei `/proc/cpuinfo` aus und erzeugt die Datei `/home/pi/icon9_51/ipl/procs/GPIO_BOARD.icn` mit dem Inhalt `$define GPIO_BOARD_xxx` (s. Tab. 25-1), die per `$include` eingeschlossen wird.

Dieses Programm erzeugt während der Laufzeit mittels `$define` globale Konstanten, die das angetroffene Board identifizieren:

$define	Modell	PCB-Revision
GPIO_BOARD_B1	B	1
GPIO_BOARD_AB2	A B	2
GPIO_BOARD_ABp2	A+ B+	1
GPIO_BOARD_CM1	Compute Module	1

Während der Code-Überarbeitung ist mir aufgefallen, dass es seitens der Anwendung sinnvoll sein könnte, von der GPIO-Library Informationen über das angetroffene Board zu erhalten. Aus diesem Grund habe ich den `initial {}`-Block der Prozedur `GPIO()` in die eigene Prozedur `GPIO_board()` „aufsteigen“ lassen.

Einen weiteren Vorteil erkennen wir spätestens jetzt in der modularen Programmierung der GPIO-Library. Wir brauchen lediglich in einer einzigen Prozedur Code zu ergänzen – und schon passt Alles für alle bisherigen RaspberryPi-Modelle – und ist entsprechend zukunftsfähig, da nur die GPIO-Layouts zukünftiger Versionen nachgetragen zu werden brauchen.

Da alle anderen Prozeduren unverändert geblieben sind, bringe ich hier nur den Code der beiden von Änderungen betroffenen Prozeduren.

/home/pi/icon9_50/bin/GPIO_BOARDdetect.icn als Lösung der Übungsaufgabe zu Kapitel 24

```
1   procedure GPIO_board()
2     if cpuinfo := open("cat /proc/cpuinfo | grep Revision | awk '{print $3}'", "p") then
3     { GPIO_REV := read(cpuinfo)
4       case GPIO_REV[-4:0] of
5       {"0002" |
6        "0003": {# Modell B PCB Revision 1.0
7                 GPIO_BOARD := "B1"
8               }
9        "0004" | # Modelle B PCB Revision 2.0
10       "0005" | # Modelle B PCB Revision 2.0
11       "0006" | # Modelle B PCB Revision 2.0
12       "0007" | # Modelle A PCB Revision 2.0
13       "0008" | # Modelle A PCB Revision 2.0
14       "0009" | # Modelle A PCB Revision 2.0
15       #"000a" |# nicht belegt
16       #"000b" |  # nicht belegt
17       #"000c" |# nicht belegt
18       "000d" | # Modelle B PCB Revision 2.0
19       "000e" | # Modelle B PCB Revision 2.0
20       "000f": { # Modelle B PCB Revision 2.0
21                 GPIO_BOARD := "AB2"
22               }
23       "0010" | # Modell B+ PCB Revision 1.0
24       "0012":  # Modell A+ PCB Revision 1.0
25               { GPIO_BOARD := "ABp1"
26               }
27       "0011":  {# Modell Compute Module PCB Revision 1.0
28                 GPIO_BOARD := "CM1"
29               }
30      }
31    }
32    if fh := open("/home/pi/icon9_51/ipl/procs/GPIO_BOARD.icn", "w") then
33    { write(fh, "$define GPIO_BOARD_" || GPIO_BOARD)
34      close(fh)
35    }
36  end
37
38  procedure main()
39    GPIO_board()
40  end
```

/home/pi/icon9_50/bin/GPIO_icn als Lösung der Übungsaufgabe zu Kapitel 24
(rote Zeilen sind geändert oder neu)

```
1   global GPIO_REV,    # Revision of Raspberry PI as produced by cpuinfo
2       GPIO_ON,   # GPIO pin set
3       GPIO_OFF,    # GPIO pin unset
4       GPIO_HIGH,   # GPIO pin set
5       GPIO_LOW,    # GPIO pin unset
6       GPIO_PINS,   # list of GPIO pins as valid for the Raspberry Pi version
7       GPIO_SEGMENT7, # displays of the 7segment display
8       GPIO_TABLE,    # table of pin status: &null, input, output
9       GPIO_BOARD   # Modell PCB Revision
...
74  procedure GPIO_board()
75    GPIO_ON  := GPIO_HIGH := 1
76    GPIO_OFF := GPIO_LOW  := 0
77    GPIO_TABLE := table()
78    if cpuinfo := open("cat /proc/cpuinfo | grep Revision | awk '{print $3}'", "p") then
79    { GPIO_REV := read(cpuinfo)
80      case GPIO_REV[-4:0] of
81      {"0002" |
82       "0003": {# Modell B PCB Revision 1.0
83              GPIO_PINS := [0,1,4,7,8,9,10,11,14,15,17,18,21,22,23,24,25]
84              GPIO_BOARD := "B1"
85            }
86       "0004" |  # Modelle B PCB Revision 2.0
87       "0005" |  # Modelle B PCB Revision 2.0
88       "0006" |  # Modelle B PCB Revision 2.0
89       "0007" |  # Modelle A PCB Revision 2.0
90       "0008" |  # Modelle A PCB Revision 2.0
91       "0009" |  # Modelle A PCB Revision 2.0
92       #"000a" | # nicht belegt
93       #"000b" |   # nicht belegt
94       #"000c" | # nicht belegt
95       "000d" |  # Modelle B PCB Revision 2.0
96       "000e" |  # Modelle B PCB Revision 2.0
97       "000f": { # Modelle B PCB Revision 2.0
98              GPIO_PINS := [2,3,4,7,8,9,10,11,14,15,17,18,22,23,24,25,27,       # P1 Header
99                                    28,29,30,31]                                 # P5 Header
100             GPIO_BOARD := "AB2"
101           }
102      "0010" |  # Modell B+ PCB Revision 1.0
103      "0012":   # Modell A+ PCB Revision 1.0
104           { GPIO_PINS := [2,3,4,5,6,7,8,9,10,11,12,13,14,15,
                             16,17,18,19,20,21,22,23,24,25,26,27]    # J8 Header
105             GPIO_BOARD := "AB+1"
106           }
107      "0011":   {# Modell Compute Module PCB Revision 1.0
108             GPIO_PINS := [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
                             16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,
                             31,32,33,34,35,36,37,38,39,40,41,42,43,44,45]
109             GPIO_BOARD := "CM1"
110           }
111     }
112     every i := 1 to *GPIO_PINS do
113     { GPIO_TABLE[GPIO_PINS[i]] := gpio_rec(GPIO_PINS[i], "", 0)
114     }
115   }
116   return GPIO_REV
117 end
118
119 procedure GPIO(n, v, d)
120   initial
```

```
121   { if /GPIO_TABLE then GPIO_board()
122   }
123
124   if /v then #v = &null => read GPIO
125   { if member(GPIO_TABLE, n) then
126     { if GPIO_TABLE[n].direction ~== "in" then GPIO_direct_to_read(n)
127       v := GPIO_read(n)
128       return v
129     }
130     else stop("GPIO " || n || " is not defined for Raspberry Pi Version ",
131           GPIO_REV, "!")
132   }
133   else# write GPIO
134   { if member (GPIO_TABLE, n) then
135     { if GPIO_TABLE[n].direction ~== "out" then GPIO_direct_to_write(n)
136       GPIO_write(n, v)
137       if \d then delay(d)
138     }
139     else stop("GPIO " || n || " is not defined for Raspberry Pi Version ",
140           GPIO_REV, "!")
141   }
142 end
```

Die Ergänzungen im GPIO-Kontrollzentrum bestehen nur in der Erweiterung der Buttons für die GPIOs und der zugehörigen Text-Eingabefelder. Hierzu werden die letzten Buttons kopiert und die Koordinaten manuell angepasst, um die Buttonliste mit gleichbleibenden Abständen fortzuführen. Beachtet aber meine Warnung in einem früheren Tutorial: Eine GUI-Anwendung funktioniert nur bei einer ordnungsgemäßen Definition der graphischen Bedienelemente – jeder Fehler resultiert in einem nicht funktionsfähigen Code.

Danach werden die Buttons und Text-Eingabefelder, die nur für bestimmte GPIO-Layouts existieren, behandelt. Auch hier habe ich eine interessante Programmiertechnik eingesetzt. Innerhalb der Routine `ui()` werden die Präprozessor-Kommandos `$ifdef` und das abschließende `$endif` eingesetzt. Diese fragen die von dem Tool `GPIO_BOARDdetect.icn` vergebene Konstante ab, die für das angetroffene Board vergeben wurde. Entsprechend des vorgefundenen Modells werden die dafür vorgesehenen Schaltflächen für die GPIO-Pins dargestellt und beschriftet.

Dann brauchen wir nur noch die farbliche Darstellung vorzunehmen.

In einer Schleife unmittelbar nach Öffnen des Fenster des GPIO-Kontrollzentrums wird die Beschriftung der Schhaltflächen abgefragt. Wann immer diese `5V`, `3V3` oder `GND` lautet, wird die Schaltfläche mit der gewünschten Hintergrundfarbe dargestellt. Die Schaltflächen, die für GPIO vorgesehen sind, verbleiben in der Hintergrundfarbe. Die Hintergrundfarbe der GPIO-Schaltflächen ändert sich erst dann nach grün, wenn die Schaltfläche gedrückt und der GPIO-Pin aktiviert wurde. Dies wird in den Callback-Routinen erledigt.

GPIO-Pins, die in den unterschiedlichen Board-Revisionen unterschiedlich belegt sind, werden je nach Board-Revision in den Calback-Funktionen unterschiedlich ausgewertet. Dabei kann es vorkommen, dass ein Button je nach Board-Revision unterschiedliche Bezeichnungen erhält. Je nach Funktion wird diese realisiert.

Auf diese Weise bleibt die GPIO-Kommandozentrale abwärts-kompatibel und kann recht leicht auf beliebige zukünftige Funktionalitäten angepasst werden.

/home/pi/icon9_50/bin/GPIO_control.icn Lösung der Übungsaufgabe zu Kapitel 24

```
1  ############################################################################
2  #
3  #   Subject:  Program to realize a GPIO Control Center
4  #
5  #   File:     /home/pi/icon9_51/bin/GPIO_control.icn
6  #
7  #   Author:   Andreas Schulz (IVQS)
8  #
9  #   Date:     January 08, 2015
10 #
11 ############################################################################
12 #
13 #
14 #
15 ############################################################################
16 #
17 #  Requires:
18 #
19 ############################################################################
20 #
21 #  Links: vsetup
22 #
23 ############################################################################
24
25 #  This vib interface specification is a working program that responds
26 #  to vidget events by printing messages.  Use a text editor to replace
27 #  this skeletal program with your own code.  Retain the vib section at
28 #  the end and use vib to make any changes to the interface.
29
30 link vsetup, vbuttons
31
32 $include "GPIO.icn"
33 $include "GPIO_BOARD.icn"
34
35 global vidgets,
36        blink_delay
37
38 procedure main(args)
39    local root, paused
40
41    (WOpen ! ui_atts()) | stop("can't open window")
42    vidgets := ui()                      # set up vidgets
43    root := vidgets["root"]
44
45    every i := 1 to 40 do
46    {  if \vidgets["button" || i] then
47       {      case vidgets["button" || i].s of
48              {"3V3":             {Bg("red");         draw_Vbutton(vidgets["button" || i])}
49              "5V":         {Bg("purple");      draw_Vbutton(vidgets["button" || i])}
50              "GND":        {Bg("gray"); draw_Vbutton(vidgets["button" || i])}
51              }
52       }
53    }
54    Bg("pale blue")
55    vidgets["button11"][8] := 1
56    every GPIO(![14,15,18,23,24,25,8,7], 1,5); delay(500)
57    every GPIO(![14,15,18,23,24,25,8,7],0); delay(500)
58    every SEGMENT7(!["9","8","7","6","5","4","3","2","1","0"], 250)
59    every SEGMENT7(!["1","2","3","4","5","6","7","8","9","0"], 250)
60    every SEGMENT7(!["A","C","E","F","H","L","P","U", "b","c","d","h","l","o","u"], 250)
61    sbar_cb1(vidgets["sbar1"], 250)
```

```
62
63      repeat
64      {
65          # handle any events that are available, or
66          # wait for events if there is no other work to do
67
68          while (*Pending() > 0) do
69          {       ProcessEvent(root, QuitCheck)
70          }
71          delay(250)
72      }
73  end
74
75  procedure button_alloff(vidget, value)
76      every i := 1 to *GPIO_PINS do
77      {   GPIO(GPIO_PINS[i], 0)
78          VSetState(vidgets["text_input" || GPIO_PINS[i]], 0)
79      }
80  end
81
82  procedure button_cb1(vidget, value)
83     return
84  end
85
86  procedure button_cb10(vidget, value)
87     /value := 0;     GPIO(15, value); VSetState(vidgets["text_input15"], value)
88     colorButton(vidgets["button10"], value, "green")
89  end
90
91  procedure button_cb11(vidget, value)
92     /value := 0;     GPIO(17, value); VSetState(vidgets["text_input17"], value)
93     colorButton(vidgets["button11"], value, "green")
94  end
95
96  procedure button_cb12(vidget, value)
97     /value := 0;     GPIO(18, value); VSetState(vidgets["text_input18"], value)
98     colorButton(vidgets["button12"], value, "green")
99  end
100
101 procedure button_cb13(vidget, value)
102    /value := 0;     GPIO(27, value); VSetState(vidgets["text_input27"], value)
103    colorButton(vidgets["button13"], value, "green")
104 end
105
106 procedure button_cb14(vidget, value)
107    return
108 end
109
110 procedure button_cb15(vidget, value)
111    /value := 0;     GPIO(22, value); VSetState(vidgets["text_input22"], value)
112    colorButton(vidgets["button15"], value, "green")
113 end
114
115 procedure button_cb16(vidget, value)
116    /value := 0;     GPIO(23, value); VSetState(vidgets["text_input23"], value)
117    colorButton(vidgets["button16"], value, "green")
118 end
119
120 procedure button_cb17(vidget, value)
121    return
122 end
123
```

```
124 procedure button_cb18(vidget, value)
125    /value := 0;     GPIO(24, value); VSetState(vidgets["text_input24"], value)
126    colorButton(vidgets["button18"], value, "green")
127 end
128 
129 procedure button_cb19(vidget, value)
130    /value := 0;     GPIO(10, value); VSetState(vidgets["text_input10"], value)
131    colorButton(vidgets["button19"], value, "green")
132 end
133 
134 procedure button_cb2(vidget, value)
135    return
136 end
137 
138 procedure button_cb20(vidget, value)
139    return
140 end
141 
142 procedure button_cb21(vidget, value)
143    /value := 0;     GPIO(9, value); VSetState(vidgets["text_input9"], value)
144    colorButton(vidgets["button21"], value, "green")
145 end
146 
147 procedure button_cb22(vidget, value)
148    /value := 0;     GPIO(25, value); VSetState(vidgets["text_input25"], value)
149    colorButton(vidgets["button22"], value, "green")
150 end
151 
152 procedure button_cb23(vidget, value)
153    /value := 0;     GPIO(11, value); VSetState(vidgets["text_input11"], value)
154    colorButton(vidgets["button23"], value, "green")
155 end
156 
157 procedure button_cb24(vidget, value)
158    /value := 0;     GPIO(8, value); VSetState(vidgets["text_input8"], value)
159    colorButton(vidgets["button24"], value, "green")
160 end
161 
162 procedure button_cb25(vidget, value)
163    return
164 end
165 
166 procedure button_cb26(vidget, value)
167    /value := 0;     GPIO(7, value); VSetState(vidgets["text_input7"], value)
168    colorButton(vidgets["button26"], value, "green")
169 end
170 
171 procedure button_cb27(vidget, value)
172    return
173 end
174 
175 procedure button_cb28(vidget, value)
176    return
177 end
178 
179 procedure button_cb29(vidget, value)
180    if vidget.s == "GPIO5" then
181    {  /value := 0
182       GPIO(5, value)
183       VSetState(vidgets["text_input29"], value)
184       colorButton(vidgets["button29"], value, "green")
185    }
```

```
186     if vidget.s == "GPIO28" then
187     {   /value := 0
188         GPIO(28, value)
189         VSetState(vidgets["text_input29"], value)
190         colorButton(vidgets["button29"], value, "green")
191     }
192 end
193 
194 procedure button_cb3(vidget, value, d)
195     # DUO-LED in GPIO2 und GPIO3, zuerst die andere GPIO-Leitung ausschalten
196     if image(VGetState(vidgets["text_input3"])) == "1" then
197     {   vidgets["button5"][8] := &null;  vidgets["button5"].D.draw_off(vidgets["button5"])
198         GPIO(3, 0)
199         VSetState(vidgets["text_input3"], 0)
200         VSetState(vidgets["button5"], &null)
201     }
202     if vidget === vidgets["blink2_3"] then
203     {   if /value then vidgets["button3"].D.draw_off(vidgets["button3"])
204         else vidgets["button3"].D.draw_on(vidgets["button3"])
205     }
206     /value := 0
207     VSetState(vidgets["text_input2"], value)
208     GPIO(2, value, d)
209    colorButton(vidgets["button3"], value, "green")
210 end
211 
212 procedure button_cb30(vidget, value)
213     if vidget.s == "GND" then return
214     if vidget.s == "GPIO29" then
215     {   /value := 0
216         GPIO(29, value)
217         VSetState(vidgets["text_input30"], value)
218         colorButton(vidgets["button30"], value, "green")
219     }
220 end
221 
222 procedure button_cb31(vidget, value)
223     if vidget.s == "GPIO6" then
224     {   /value := 0
225         GPIO(6, value);
226         VSetState(vidgets["text_input31"], value)
227         colorButton(vidgets["button31"], value, "green")
228     }
229     if vidget.s == "GPIO30" then
230     {   /value := 0
231         GPIO(30, value);
232         SetState(vidgets["text_input31"], value)
233         colorButton(vidgets["button31"], value, "green")
234     }
235 end
236 
237 procedure button_cb32(vidget, value)
238    if vidget.s == "GPIO12" then
239     {   /value := 0
240         GPIO(12, value)
241         VSetState(vidgets["text_input32"], value)
242         colorButton(vidgets["button32"], value, "green")
243     }
244     if vidget.s == "GPIO31" then
245     {   /value := 0
246         GPIO(31, value)
247         VSetState(vidgets["text_input32"], value)
```

```
248         colorButton(vidgets["button32"], value, "green")
249     }
250 end
251 
252 procedure button_cb33(vidget, value)
253     if vidget.s == "GPIO13" then
254     {   /value := 0
255         GPIO(13, value)
256         VSetState(vidgets["text_input33"], value)
257         colorButton(vidgets["button33"], value, "green")
258     }
259     if vidget.s == "GND" then return
260 end
261 
262 procedure button_cb34(vidget, value)
263    return
264 end
265 
266 procedure button_cb35(vidget, value)
267    /value := 0;     GPIO(19, value); VSetState(vidgets["text_input35"], value)
268    colorButton(vidgets["button35"], value, "green")
269 end
270 
271 procedure button_cb36(vidget, value)
272    /value := 0;     GPIO(16, value); VSetState(vidgets["text_input36"], value)
273    colorButton(vidgets["button36"], value, "green")
274 end
275 
276 procedure button_cb37(vidget, value)
277    /value := 0;     GPIO(26, value); VSetState(vidgets["text_input37"], value)
278    colorButton(vidgets["button37"], value, "green")
279 end
280 
281 procedure button_cb38(vidget, value)
282    /value := 0;     GPIO(20, value); VSetState(vidgets["text_input38"], value)
283    colorButton(vidgets["button38"], value, "green")
284 end
285 
286 procedure button_cb39(vidget, value)
287    return
288 end
289 
290 procedure button_cb4(vidget, value)
291    return
292 end
293 
294 procedure button_cb40(vidget, value)
295    /value := 0;     GPIO(21, value); VSetState(vidgets["text_input40"], value)
296    colorButton(vidgets["button40"], value, "green")
297 end
298 
299 procedure button_cb5(vidget, value, d)
300     # DUO-LED in GPIO2 und GPIO3, zuerst die andere GPIO-Leitung ausschalten
301     if image(VGetState(vidgets["text_input2"])) == "1" then
302     {   vidgets["button3"][8] := &null;  vidgets["button3"].D.draw_off(vidgets["button3"])
303         GPIO(2, 0)
304         VSetState(vidgets["text_input2"], 0)
305         VSetState(vidgets["button3"], &null)
306     }
307     if vidget === vidgets["blink2_3"] then
308     {   if /value then vidgets["button5"].D.draw_off(vidgets["button5"])
309         else vidgets["button5"].D.draw_on(vidgets["button5"])
```

```
310     }
311     /value := 0
312     VSetState(vidgets["text_input3"], value)
313     GPIO(3, value, d)
314     colorButton(vidgets["button5"], value, "green")
315 end
316 
317 procedure button_cb6(vidget, value)
318     return
319 end
320 
321 procedure button_cb7(vidget, value)
322     /value := 0
323     GPIO(4, value)
324     VSetState(vidgets["text_input4"], value)
325     colorButton(vidgets["button7"], value, "green")
326 end
327 
328 procedure button_cb8(vidget, value)
329     /value := 0
330     GPIO(14, value)
331     VSetState(vidgets["text_input14"], value)
332     colorButton(vidgets["button8"], value, "green")
333 end
334 
335 procedure button_cb9(vidget, value)
336     return
337 end
338 
339 procedure blink2_3(vidget, value)
340     if /blink_delay then sbar_cb1(vidgets["sbar1"])
341     every i := 1 to 10 do
342     {   button_cb3(vidget, GPIO_ON, blink_delay)
343         button_cb3(vidget, GPIO_OFF, blink_delay)
344         button_cb5(vidget, GPIO_ON, blink_delay)
345         button_cb5(vidget, GPIO_OFF, blink_delay)
346     }
347 end
348 
349 procedure sbar_cb1(vidget, value)
350     blink_delay := value
351 end
352 
353 procedure button_quit(vidget, value)
354     GPIO_unexport(GPIO_TABLE)
355     exit()
356 end
357 
358 
359 procedure SegmentA_cb(videt, value)
360     if /value then SEGMENT7_off("A") else SEGMENT7_on("A")
361     colorButton(vidgets["SegmentA"], value, "yellow")
362 end
363 
364 procedure SegmentB_cb(videt, value)
365     if /value then SEGMENT7_off("B") else SEGMENT7_on("B")
366     colorButton(vidgets["SegmentB"], value, "yellow")
367 end
368 
369 procedure SegmentC_cb(videt, value)
370     if /value then SEGMENT7_off("C") else SEGMENT7_on("C")
371     colorButton(vidgets["SegmentC"], value, "yellow")
```

```
372 end
373
374 procedure SegmentD_cb(videt, value)
375    if /value then SEGMENT7_off("D") else SEGMENT7_on("D")
376    colorButton(vidgets["SegmentD"], value, "yellow")
377 end
378
379 procedure SegmentE_cb(videt, value)
380    if /value then SEGMENT7_off("E") else SEGMENT7_on("E")
381    colorButton(vidgets["SegmentE"], value, "yellow")
382 end
383
384 procedure SegmentF_cb(videt, value)
385    if /value then SEGMENT7_off("F") else SEGMENT7_on("F")
386    colorButton(vidgets["SegmentF"], value, "yellow")
387 end
388
389 procedure SegmentG_cb(videt, value)
390    if /value then SEGMENT7_off("G") else SEGMENT7_on("G")
391    colorButton(vidgets["SegmentG"], value, "yellow")
392 end
393
394 procedure SegmentP_cb(videt, value)
395    if /value then SEGMENT7_off("P") else SEGMENT7_on("P")
396    colorButton(vidgets["Segmentp"], value, "yellow")
397 end
398
399 procedure TI_7Segment_cb(vidget, value)
400    SEGMENT7_off("ABCDEFGP")
401    SEGMENT7(value)
402 end
403
404 procedure colorButton(vid, val, col)
405    if \val then
406    {  if val = 1 then
407       {      alt_bg := Bg()
408              Bg(col)
409              draw_Vbutton(vid)
410       }
411    }
412    Bg(\alt_bg)
413 end
414
415 #===<<vib:begin>>===      modify using vib; do not remove this marker line
416 procedure ui_atts()
417     return ["size=597,800", "label=GPIO_control V2 \xa9 08-JAN-2015 by Andreas Schulz",
418 "bg=pale blue"]
419 end
420
421 procedure ui(win, cbk)
422    return vsetup(win, cbk,
423    [":Sizer:::0,0,597,800:",],
424    ["SegmentA:Button:regular:1:437,21,75,17:",SegmentA_cb],
425    ["SegmentB:Button:regular:1:512,39,15,75:",SegmentB_cb],
426    ["SegmentC:Button:regular:1:512,128,15,75:",SegmentC_cb],
427    ["SegmentD:Button:regular:1:437,203,75,17:",SegmentD_cb],
428    ["SegmentE:Button:regular:1:420,128,15,75:",SegmentE_cb],
429    ["SegmentF:Button:regular:1:420,39,15,75:",SegmentF_cb],
430    ["SegmentG:Button:regular:1:437,114,75,17:",SegmentG_cb],
431    ["Segmentp:Button:regular:1:528,203,17,17:",SegmentP_cb],
432    ["TI_7Segment:Text::1:442,258,80,19:7Segment:\\=",TI_7Segment_cb],
433    ["alloff:Button:regular::181,666,106,48:ALL OFF",button_alloff],
```

```
434     ["blink2_3:Button:check::63,61,62,50:Blink",blink2_3],
435     ["button1:Button:regular:1:179,23,50,30:3V3",button_cb1],
436     ["button10:Button:regular:1:238,151,50,30:GPIO15",button_cb10],
437     ["button11:Button:regular:1:179,183,50,30:GPIO17",button_cb11],
438     ["button12:Button:regular:1:238,183,50,30:GPIO18",button_cb12],
439     ["button13:Button:regular:1:179,215,50,30:GPIO27",button_cb13],
440     ["button14:Button:regular:1:238,215,50,30:GND",button_cb14],
441     ["button15:Button:regular:1:179,247,50,30:GPIO22",button_cb15],
442     ["button16:Button:regular:1:238,247,50,30:GPIO23",button_cb16],
443     ["button17:Button:regular:1:179,279,50,30:3V3",button_cb17],
444     ["button18:Button:regular:1:238,279,50,30:GPIO24",button_cb18],
445     ["button19:Button:regular:1:179,312,50,30:GPIO10",button_cb19],
446     ["button2:Button:regular:1:238,23,50,30:5V",button_cb2],
447     ["button20:Button:regular:1:238,312,50,30:GND",button_cb20],
448     ["button21:Button:regular:1:179,344,50,30:GPIO9",button_cb21],
449     ["button22:Button:regular:1:238,344,50,30:GPIO25",button_cb22],
450     ["button23:Button:regular:1:179,376,50,30:GPIO11",button_cb23],
451     ["button24:Button:regular:1:238,376,50,30:GPIO8",button_cb24],
452     ["button25:Button:regular:1:179,408,50,30:GND",button_cb25],
453     ["button26:Button:regular:1:238,408,50,30:GPIO7",button_cb26],
454     ["button3:Button:regular:1:179,55,50,30:GPIO2",button_cb3],
455     ["button4:Button:regular:1:238,55,50,30:5V",button_cb4],
456     ["button5:Button:regular:1:179,87,50,30:GPIO3",button_cb5],
457     ["button6:Button:regular:1:238,87,50,30:GND",button_cb6],
458     ["button7:Button:regular:1:179,119,50,30:GPIO4",button_cb7],
459     ["button8:Button:regular:1:238,119,50,30:GPIO14",button_cb8],
460     ["button9:Button:regular:1:179,151,50,30:GND",button_cb9],
461 $ifdef GPIO_BOARD_Abp1
462     ["button27:Button:regular:1:179,440,50,30:ID_SD",button_cb27],
463     ["button28:Button:regular:1:238,440,50,30:ID_SC",button_cb28],
464     ["button29:Button:regular:1:179,472,50,30:GPIO5",button_cb29],
465     ["button30:Button:regular:1:238,472,50,30:GND",button_cb30],
466     ["button31:Button:regular:1:179,504,50,30:GPIO6",button_cb31],
467     ["button32:Button:regular:1:238,504,50,30:GPIO12",button_cb32],
468     ["button33:Button:regular:1:179,536,50,30:GPIO13",button_cb33],
469     ["button34:Button:regular:1:238,536,50,30:GND",button_cb34],
470     ["button35:Button:regular:1:179,568,50,30:GPIO19",button_cb35],
471     ["button36:Button:regular:1:238,568,50,30:GPIO16",button_cb36],
472     ["button37:Button:regular:1:179,600,50,30:GPIO26",button_cb37],
473     ["button38:Button:regular:1:238,600,50,30:GPIO20",button_cb38],
474     ["button39:Button:regular:1:179,632,50,30:GND",button_cb39],
475     ["button40:Button:regular:1:238,632,50,30:GPIO21",button_cb40],
476 $endif
477 $ifdef GPIO_BOARD_AB2
478     ["button27:Button:regular:1:179,440,50,30:5V",button_cb27],
479     ["button28:Button:regular:1:238,440,50,30:3V3",button_cb28],
480     ["button29:Button:regular:1:179,472,50,30:GPIO28",button_cb29],
481     ["button30:Button:regular:1:238,472,50,30:GPIO39",button_cb30],
482     ["button31:Button:regular:1:179,504,50,30:GPIO30",button_cb31],
483     ["button32:Button:regular:1:238,504,50,30:GPIO31",button_cb32],
484     ["button33:Button:regular:1:179,536,50,30:GND",button_cb33],
485     ["button34:Button:regular:1:238,536,50,30:GND",button_cb34],
486 $endif
487     ["label1:Label:::471,5,7,13:A",],
488     ["label10:Label:::346,157,7,13:B",],
489     ["label11:Label:::346,188,7,13:C",],
490     ["label12:Label:::345,254,7,13:D",],
491     ["label13:Label:::345,285,7,13:E",],
492     ["label14:Label:::345,350,7,13:F",],
493     ["label15:Label:::346,383,7,13:G",],
494     ["label16:Label:::347,416,7,13:P",],
495     ["label2:Label:::531,64,7,13:B",],
```

```
496     ["label3:Label:::531,158,7,13:C",],
497     ["label4:Label:::470,222,7,13:D",],
498     ["label5:Label:::407,157,7,13:E",],
499     ["label6:Label:::407,67,7,13:F",],
500     ["label7:Label:::473,97,7,13:G",],
501     ["label8:Label:::533,222,7,13:P",],
502     ["label9:Label:::348,127,7,13:A",],
503     ["line1:Line:::233,19,233,666:",],
504     ["menu1:Menu:pull::8,2,71,21:Direction",menu_cb1,
505        ["GPIO > GND","3V3  > GPIO"]],
506     ["quit:Button:regular::179,718,106,48:QUIT",button_quit],
507     ["sbar1:Scrollbar:h:1:5,53,172,10:0,1000,250",sbar_cb1],
508     ["text_input10:Text::6:124,315,52,19:\\=",text_input_cb10],
509     ["text_input11:Text::6:123,381,52,19:\\=",text_input_cb11],
510     ["text_input14:Text::6:289,123,52,19:\\=",text_input_cb14],
511     ["text_input15:Text::6:288,155,52,19:\\=",text_input_cb15],
512     ["text_input17:Text::6:124,187,52,19:\\=",text_input_cb17],
513     ["text_input18:Text::6:288,187,52,19:\\=",text_input_cb18],
514     ["text_input2:Text::6:124,65,52,19:\\=",text_input_cb2],
515     ["text_input22:Text::6:123,252,52,19:\\=",text_input_cb22],
516     ["text_input23:Text::6:288,251,52,19:\\=",text_input_cb23],
517     ["text_input24:Text::6:289,283,52,19:\\=",text_input_cb24],
518     ["text_input25:Text::6:288,348,52,19:\\=",text_input_cb25],
519     ["text_input3:Text::6:125,87,52,19:\\=",text_input_cb3],
520     ["text_input4:Text::6:125,124,52,19:\\=",text_input_cb4],
521     ["text_input7:Text::6:288,413,52,19:\\=",text_input_cb7],
522     ["text_input8:Text::6:289,381,52,19:\\=",text_input_cb8],
523     ["text_input9:Text::6:124,349,52,19:\\=",text_input_cb9],
524 $ifdef GPIO_BOARD_Abp1
525     ["text_input27:Text::6:123,220,52,19:\\=",text_input_cb27],
526     ["text_input29:Text::6:123,472,52,19:\\=",text_input_cb29],
527     ["text_input31:Text::6:123,504,52,19:\\=",text_input_cb31],
528     ["text_input32:Text::6:288,504,52,19:\\=",text_input_cb32],
529     ["text_input33:Text::6:123,536,52,19:\\=",text_input_cb33],
530     ["text_input35:Text::6:123,568,52,19:\\=",text_input_cb35],
531     ["text_input36:Text::6:288,568,52,19:\\=",text_input_cb36],
532     ["text_input37:Text::6:123,600,52,19:\\=",text_input_cb37],
533     ["text_input38:Text::6:288,600,52,19:\\=",text_input_cb38],
534     ["text_input40:Text::6:288,632,52,19:\\=",text_input_cb40],
535 $endif
536  )
537 end
538 #===<<vib:end>>===  end of section maintained by vib
539
540
```

25 Ereignisbehandlung bei Anwendungen mit mehr als einem Fenster

Alles bisher Besprochene gilt nur für Anwendungen mit einer einzigen graphischen Benutzeroberfläche. Nach allem, was wir bisher gelernt haben, sind wir (noch) nicht wirklich in der Lage, Anwendungen mit mehreren gleichzeigtig erscheinenden Fenstern zu programmieren – geschweige denn deren Ereignisschleifen zu programmieren.

25.1 VIB-Betrachtungen

Falls Deine Anwendung mehr als eine graphische Benutzeroberfläche erfordert, dann muss jede davon in separaten Dateien implementiert werden. Nachdem VIB diese Dateien erzeugt hat, musst Du die Prozedur `main()` in allen Dateien löschen, außer im Hauptprogramm, das die graphische Benutzeroberflächen einbindet / mit diesen verbindet.

VIB-Schnittstellen-Code enthält üblicherweise

zwei Prozeduren, die per Voreinstellung `ui_atts()` and `ui()` heißen. `ui_atts()` gibt die Attribute zurück, die verwendet werden, um das Hauptfenster der Anwendung zu öffnen. Diese Prozedur kann verwendet werden, um das Fenster mit zusätzlichen Attributen zu öffnen. `ui()` öffnet das Fenster der graphischen Benutzeroberfläche, falls `&window` gleich `&null` ist, zeichnet die Vidgets und initialisiert die Benutzeroberfläche.

Pass auf, dass Du individuelle Prozedurnamen für jede der Prozeduren `ui_atts()` und `ui()` vergibst, was im Dialogfenster der VIB-Zeichenoberfläche im Text-Eingabefeld `procedure name` getan wird.

Da jede Benutzeroberfläche seinen eigenen Satz an Vidgets hat, können identische Vidget-IDs in mehreren VIB-Schnittstellen verwendet werden.

Zum Beispiel:

```
interface1_vidgets := interface1()
interface1_win := &window
&window := &null
interface2 := interface2()
interface2_win := &window
```

oder

```
interface1_win := WOpen! interface_atts()
interface1_vidgets := interface(interface1_win)
interface2_win := WOpen ! interface2_atts_win)
interface2_vidgets := interface(interface2_win)
```

25.2 Mehrfache VIB-basierte Benutzeroberflächen steuern

Verwende folgenden Code, um das root-Vidget für jede Benutzeroberfläche zu erhalten:

```
interface1_root := interface1_vidgets["root"]
interface2_root := interface2_vidgets["root"]
```

Hier folgt die Ereignisschleife für eine Ereignisgetriebene Anwendung

```
repeat
{  root := case Active() of
   {  interface1_win := interface1_root
      interface2_win := interface2_root
   }
   ProcessEvent(root, …)
}
```

anderenfalls:

```
root := interface1_root
while ProcessEvent(root, …)
…
```

Code-Ausschnitt aus einem VIB-Schnittstellencode (interface1.icn)

```
procedure go_interface2()
   while get(Pending(interface2_win))
   root := interface2_root
end
```

Code-Ausschnitt aus einem VIB-Schnittstellencode (interface2.icn)

```
procedure go_interface1()
   while get(Pending(interface1_win))
   root := interface1_root
end
```

Die Ereignisbehandlungsschleife:

```
repeat
{  while *Pending(interface1_win |
                  interface2_win) > 0 do
   {  ProcessEvent(
      case Active() of
      { interface1_win: interface1_root
        interface2_win: interface2_root
      },
      …
      )
      # do something before checking the
next event
      …
   }
}
```

25.3 Praktisches Beispiel

25.3.1 Beschreibung der Anwendung

Wir wollen eine einfache Anwendung erstellen, die zwei Fenster anzeigt, die mit VIB kreiert werden. Das Hauptprogramm soll im Wesentlichen nur aus einer Ereignisbehandlungsschleife bestehen, die je nach Ereignis entsprechende Prozeduren aufruft. Auf jedem der beiden Fenster erscheint ein Listenfeld und zwei Schaltflächen. Anklicken der Schaltfläche `Quit` beendet das Programm, die andere Schaltfläche `Time => Fenster2` bzw. `Time >= Fenster1` bewirkt, dass die aktuelle Uhrzeit in das Listenfeld des jeweils anderen Fensters eingetragen wird.

25.3.2 VIB: Erstellen von Fenster 1

Als Erstes erstellen wir mit VIB die graphsiche Benutzeroberfläche für das erste Fenster `Win1`. Dazu starten wir VIB und klicken mit der rechten Maustaste das kleine Quadrat der zukünfigen Fensterumrandung an und öffnen damit das Dialogfenster der VIB-Zeichenoberfläche (s. Abbildung 25-1).

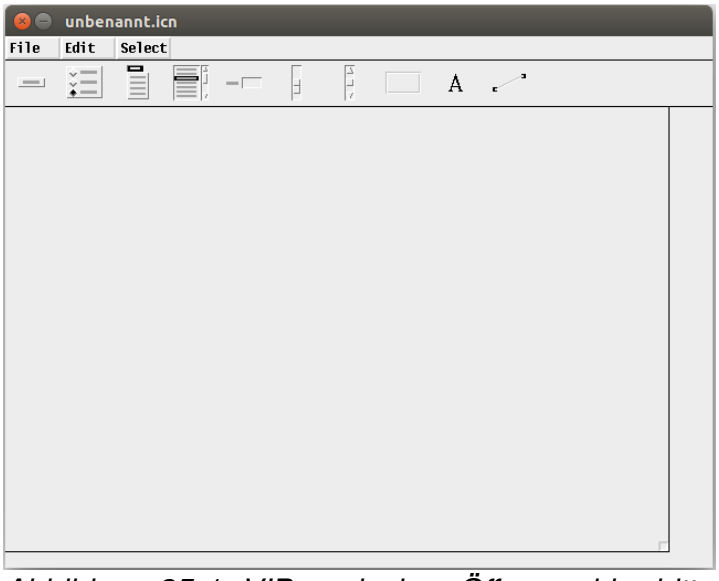

Abbildung 25-1: VIB nach dem Öffnen – hier bitte das kleine Quadrat rechts unten mit der rechten Maustaste anklicken

Wir geben die in Abbildung 25-2 enthaltenen Informationen ein und klicken die Schaltfläche `Okay` an.

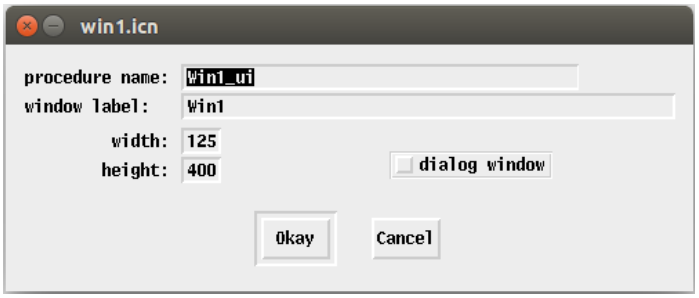

Abbildung 25-2: Benennen der Prozedur, die das User-Interface (`ui`) bereitstellt, Benennung des Fensters sowie Definition der Breite und der Höhe des Fensters

25.3.3 VIB: Erzeugen des Listenfeldes

Danach ziehen wir das Symbol für ein Listenfeld auf die VIB-Zeichenoberfläche. Das Listenfeld klicken wir mit der rechten Maustaste an und geben die in der folgenden Abbildung enthaltenen Informationen ein und definieren auf diese Weise Inhalt, Aussehen, Position und Größe des Listenfeldes sowie dessen Bezeichnung, ID und Callback-Funktion.

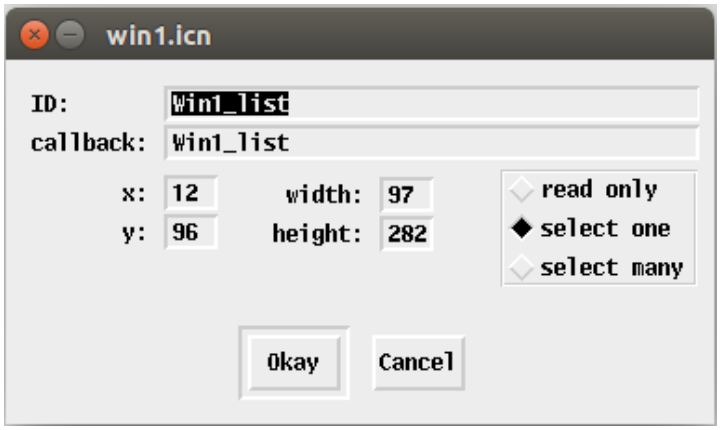

Abbildung 25-3: Parameter für das Listenfeld

25.3.4 VIB: Erzeugen der Schaltfläche `QUIT`

Danach ziehen wir das Symbol für eine Schaltfläche auf die VIB-Zeichenoberfläche. Die Schaltfläche klicken wir mit der rechten Mausaste an und geben die in der folgenden Abbildung enthaltenen Informationen ein und definieren auf diese Weise Inhalt, Aussehen, Position und Größe der Schaltfläche sowie deren Bezeichnung, ID und Callback-Funktion.

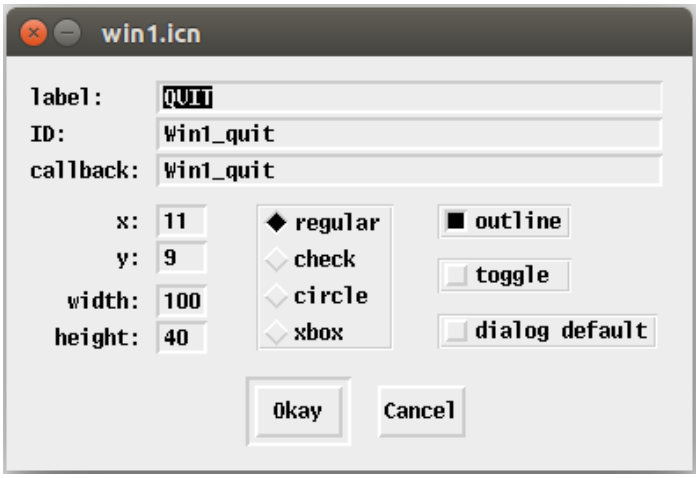

Abbildung 25-4: Parameter für die Schaltfläche `QUIT`

25.3.5 VIB: Erzeugen der Schaltfläche `Time...`

Danach ziehen wir das Symbol für eine Schaltfläche auf die VIB-Zeichenoberfläche. Die Schaltfläche klicken wir mir der rechten Maustaste an und geben die in der folgenden Abbildung enthaltenen Informationen ein und definieren auf diese Weise Inhalt, Aussehen, Position und Größe der Schaltfläche sowie deren Bezeichnung, ID und Callback-Funktion.

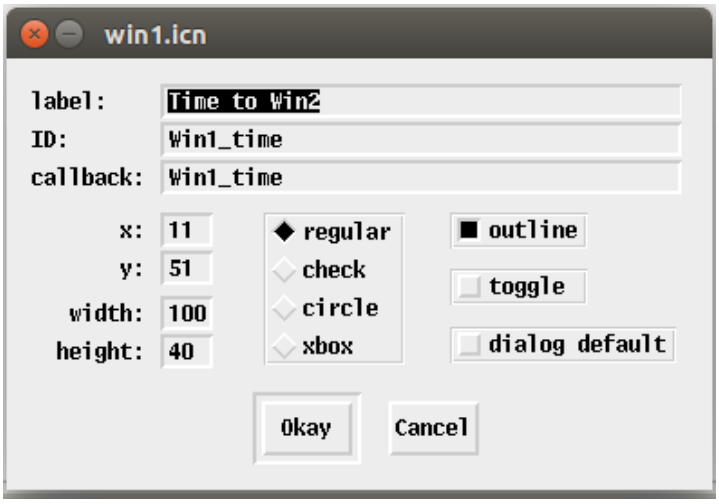

Abbildung 25-5: Parameter für die Schaltfläche `Time`

25.3.6 Quellcode der Anwendung für `win1.icn`

Wir gehen ins VIB-Menü `File` und wählen das Untermenü `Save as` aus. Daraufhin erscheint ein Text-Eingabefeld, in das wir folgenden Pfad eintragen:

`/home/pi/icon9_51/bin/win1.icn`

Danach verlassen wir VIB, indem wir im Menü `File` den Eintrag `Quit` auswählen.

Danach öffnen wir unseren bevorzugten Editor bzw. IDE und lassen das Programm einmal compilieren und ablaufen.

Der Quellcode sieht dann ziemlich so wie der hier aus:

/home/pi/icon9_50/bin/win1.icn

```
1   ############################################################################
2   #
3   #  File:     /home/andreas/icon9_51/bin/win1.icn
4   #
5   #  Subject:  Program to ...
6   #
7   #  Author:
8   #
9   #  Date:     January 14, 2015
10  #
11  ############################################################################
12  #
13  #
14  #
15  ############################################################################
16  #
17  #  Requires:
18  #
19  ############################################################################
20  #
21  #  Links: vsetup
22  #
23  ############################################################################
24
25  #  This vib interface specification is a working program that responds
26  #  to vidget events by printing messages.  Use a text editor to replace
27  #  this skeletal program with your own code.  Retain the vib section at
28  #  the end and use vib to make any changes to the interface.
29
30  link vsetup
31
32  procedure main(args)
33
34     (WOpen ! ui_atts()) | stop("can't open window")
35     vidgets := ui()                 # set up vidgets
36     root := vidgets["root"]
37
38     paused := 1              # flag no work to do
39     repeat {
40        # handle any events that are available, or
41        # wait for events if there is no other work to do
42        while (*Pending() > 0) | \paused do {
43           ProcessEvent(root, QuitCheck)
44           }
45        # if <paused> is set null, code can be added here
46        # to perform useful work between checks for input
47        }
48  end
49
50  procedure Win1_quit(vidget, value)
51     return
52  end
53
```

Abbildung 25-6: Screenshot `Win1`

```
54  procedure Win1_time(vidget, value)
5      return
56  end
57
58  procedure Win1_list(vidget, value)
59     return
60  end
61
62  #===<<vib:begin>>===  modify using vib; do not remove this marker line
63  procedure Win1_ui_atts()
64     return ["size=125,400", "bg=pale gray", "label=Win1"]
65  end
66
67  procedure Win1_ui(win, cbk)
68  return vsetup(win, cbk,
69     ["Win1_ui:Sizer:::0,0,125,400:Win1",],
70     ["Win1_list:List:w::12,96,97,282:",Win1_list],
71     ["Win1_quit:Button:regular::11,9,100,40:QUIT",Win1_quit],
72     ["Win1_time:Button:regular::11,51,100,40:Time to Win2",Win1_time],
73     )
74  end
75  #===<<vib:end>>===  end of section maintained by vib
```

25.3.7 Erstellen von Fenster 2

Um das Programm für Fenster 2 zu erstellen, könnten wir analog vorgehen. Da Programmierer in der Regel aber besonders faule Menschen sind, verhalten wir uns jetzt auch mal so. Irgendwie, na ja, wer bis hierher durchgehalten hat, darf sich durchaus mal so wie richtige Programmierer verhalten. Also lasst uns zum Quellcode für Fenster 2 auf Faulenart gelangen.

1. Wir kopieren den Quellcode von `win1.icn`. Dazu geben wir in der Konsole ein

   ```
   cp /home/pi/icon9_51/bin/win1.icn
   /home/pi/icon9_51/bin/win2.icn
   ```

2. Wir starten Geany (bzw. Deinen bevorzugten Editor / IDE) und passen `win2.icn` an, indem wir den Text `Win1` durch `Win2` ersetzen. In Geany drücken wir die Tastenkombination `Strg-H`, suchen nach `Win1`, ersetzen durch `Win2`, drücken auf `Alle ersetzen`, klicken auf die Schaltfläche `Im Dokument` – und fertig!

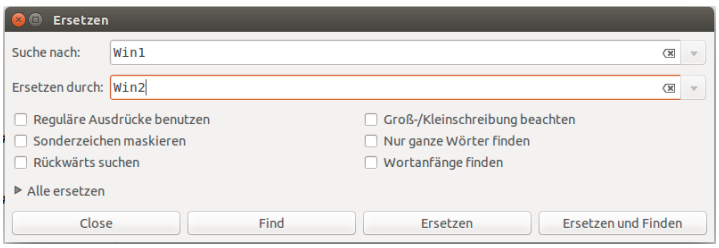

Abbildung 25-7: Suchen und Ersetzen

Dann prüfen wir mal, ob das Programm – erstellt auf die Faulenart, äh, Programmiererart – denn auch funktioniert. F5 gedrückt und schon erscheint auf dem Bildschirm:

Abbildung 25-8: Screenshot `Win2`

Huch! Jetzt waren wir zwar korrekt faul aber einen Schritt zu schnell. Wir wollen natürlich, dass im Fenster `Win2` auf der Schaltfläche `Time to Win1` erscheint. Ändert das bitte im Quellcode...

25.3.8 Erstellen des Hauptprogramms

Unser Hauptprogramm soll die beiden Quellcodes `win1.icn` und `win2.icn` einschließen. Beide Fenster sollen geöffnet werden. Wenn dies erfolgreich gelingt, dann soll eine Ereignisbehandlungsschleife abfragen, ob an irgendeinem Fenster ein Ereignis ansteht und dann die dafür vorgesehene Callback-Routine aufrufen.

25.3.8.1 Annäherung 1: Einbinden der Quellcodes

Wir geben für das Hauptprogramm `MehrereFenster.icn` erst einmal ein Minimal-Programm ein.

/home/pi/icon9_50/bin/MehrereFenster.icn

```
1  link graphics
2
3  $include "win1.icn"
4  $include "win2.icn"
```

```
5
6  procedure main()
7
8  end
```

Auch wenn wir wissen, dass das Programm noch ganz weit von einer Lösung entfernt ist, compilieren wir es einmal. Zwei Fehler erscheinen (s. Abbildung 25-9). Versuchen wir einmal, die beiden Fehlermeldungen zu verstehen – und danach abzustellen. Denn ohne Fehlermeldungen bekommen wir kein lauffähiges Programm hin.
Und je früher in der Entwicklung grundlegende Fehler eliminiert werden, um so leichter fällt dann auch die Entwicklung.
In Zeile 3 von `MehrereFenster.icn` wird `win1.icn` eingeschlossen. Das scheint fehlerfei zu compilieren – denn es kommen keine Fehlermeldungen. Warum ich hier vorsichtig sage „scheint fehlerfrei zu compilieren“ - und warum in `win1.icn` doch ein Fehler steckt, werden wir gleich verstehen.

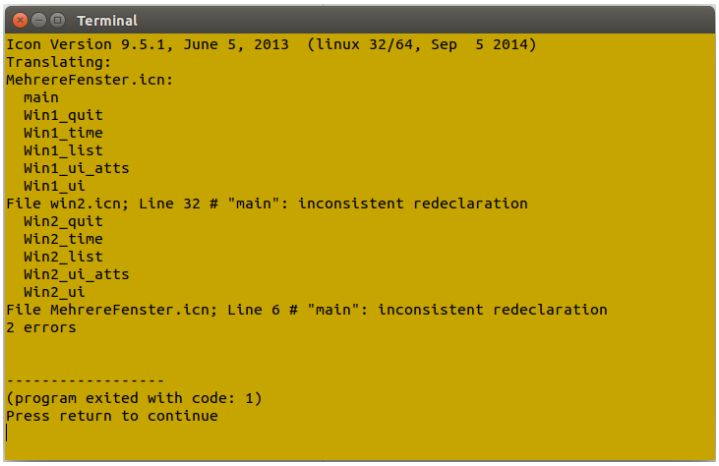

Abbildung 25-9: Screenshot mit zwei Compilierfehlern

Der erste Fehler kommt dann in `win2.icn`. Hier wird bemängelt, dass `main` inkonsistent redeklariert wird. Na klar, logisch! `MehrereFenster.icn` hat eine Prozedur `main()` - und `win1.icn` und `win2.icn` auch – denn beide lieferten ja vorhin ausführbare Programme, die ohne `main()` nicht laufen.

OK. Dann kommt der gleiche Fehler noch einmal – bezieht sich aber auf die Prozedur `main()` in `MehrereFenster.icn`.

Drei Quellprogramme, drei Prozeduren mit dem Namen `main()` - aber nur zwei Fehlermeldungen. Welche der drei Prozeduren ist nun die richtige, welche beiden müssen weichen?

Rein formal ist es vollkommen egal, welche beiden wir löschen. Da wir vorhin gesagt haben, dass `MehrereFenster.icn` das Hauptprogramm werden soll, dann finde ich, steht dem Teil auch eine Prozedur `main()` richtig gut. Die anderen beiden Prozeduren `main()` löschen wir (erst einmal) nicht sondern setzen sie in Kommentare – also ein # vor jede Zeile setzen. Und dann compilieren wir `MehrereFenster.icn` gleich nochmal.

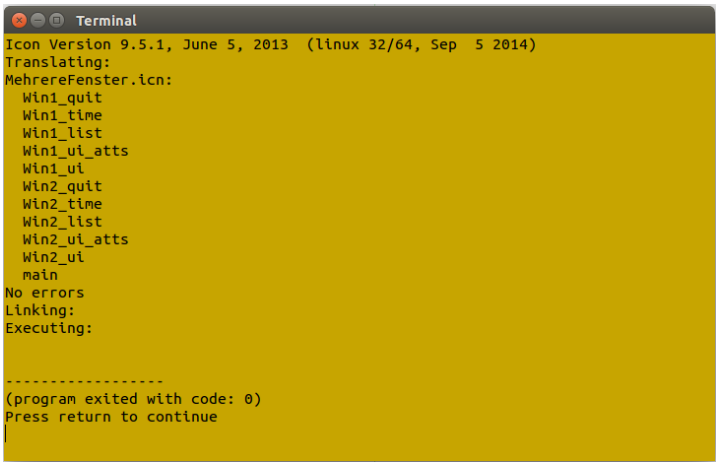

Abbildung 25-10: Screenshot ohne Compilierfehler

OK. Es passiert zwar gar nichts – aber die Compilierfehler sind weg. Ich sage mal, wir befinden uns auf einem richtigen Weg.

25.3.8.2 Annäherung 2: Fenster aus `win1.icn` und `win2.icn` öffnen

Als nächstes motzen wir den Quellcode zu `Mehrere Fenster.icn` auf. Wir versuchen als Erstes, dass die Fenster von `win1.icn` und `win2.icn` erscheinen.

/home/pi/icon9_50/bin/MehrereFenster.icn

```
1   link graphics
2
3   $include "win1.icn"
4   $include "win2.icn"
5
6   procedure main()
7     WOpen ! Win1_ui_atts()
8     WOpen ! Win2_ui_atts()
9     WDone()
10  end
```

Compilieren, starten – Die beiden Fenster von `win1.icn` und `win2.icn` erscheinen, allerdings ohne Bedienelemente. Das dürfte jetzt auch niemanden verwundern, weil die Prozeduren `Winx_ui_atts()` ja nur die Fensterparameter liefern, mit denen lediglich die Fenster geöffnet werden sollen.

OK. Die Zeilen 7 und 8 funktionieren zwar. Wir hatten aber früher einmal gelernt, dass wir mögliche Fehler abfangen sollten. Und wir wissen auch, ein einfacher Fehler bei den Fensterparametern – und schon funktioniert `WOpen()` nicht.

25.3.8.3 Annäherung 3: Fenster aus `win1.icn` und `win2.icn` mit Bedienelementen

Wir bauen in die Zeilen, in denen die Fenster `Win1` und `Win2` geöffnet werden, eine kleine Fehlerbehandlung ein. In diesem Fall wählen wir die brutalste Variante: Programmabbruch, wenn es nicht geöffnet werden kann. Dies ist hier gerechtfertigt, weil das Programm sinnlos wird, weitergeführt zu werden, sobald auch nur ein einziges Fenster nicht geöffnet wrden kann.

/home/pi/icon9_50/bin/MehrereFenster.icn

```
1   link graphics
2
3   $include "win1.icn"
4   $include "win2.icn"
5
6   procedure main()
7     Win1   :=   WOpen   !   Win1_ui_atts()   |
    stop("Fenster   Win1   kann   nicht   geöffnet
    werden! - Programmabbruch")
8     Win1_vidgets := Win1_ui()
9     Win1_root := Win1_vidgets["root"]
10    &window := &null
11
12    Win2   :=   WOpen   !   Win2_ui_atts()   |
13  stop("Fenster   Win2   kann   nicht   geöffnet
14  werden! - Programmabbruch")
15    Win2_vidgets := Win2_ui()
16    Win2_root := Win2_vidgets["root"]
17
18    WDone()
19  end
```

Jetzt sieht unser Ergebnis so aus (s. Abbildung 25-11). Klickt mal auf die Bedienelemente. OK, passiert nichts. Kann auch nicht, solange niemand die Ereignisbehandlung und die Callback-Prozeduren erdichtet hat.

25.3.8.4 Annäherung 4: Ereignisbehandlung über `ProcessEvent()`

Da es verschiedene Arten der Ereignisbehandlung gibt, stelle ich verschiedene Methoden vor. Sie sind alle gleichwertig. Probiert sie einfach aus und verwendet dann die Methodik, die Euch am meisten zusagt.

Abbildung 25-11: Screenshot mit Bedienelementen

/home/pi/icon9_51/bin/MehrereFenster.icn

```
1   link graphics
2
3   $include "win1.icn"
4   $include "win2.icn"
5
6   procedure main()
7     Win1   :=   WOpen   !   Win1_ui_atts()   |
    stop("Fenster   Win1   kann   nicht   geöffnet
    werden! - Programmabbruch")
8     Win1_vidgets := Win1_ui()
9     Win1_root := Win1_vidgets["root"]
10    &window := &null
11
12    Win2   :=   WOpen   !   Win2_ui_atts()   |
    stop("Fenster   Win2   kann   nicht   geöffnet
    werden! - Programmabbruch")
13    Win2_vidgets := Win2_ui()
14    Win2_root := Win2_vidgets["root"]
15
16    repeat
17    { while *Pending(Win1 | Win2) > 0 do
18      { ProcessEvent(  case Active() of
19                  { Win1:     Win1_root
20                    Win2:     Win2_root
21                  }
22                )
23      }
24    }
25
26    WDone()
27  end
```

25.3.8.5 Annäherung 5: Callback-Routinen einbauen

OK – die Callbacks hätten wir auch schon in `win1.icn` und `win2.icn` einbauen können. Dass wir das erst jetzt machen, entspricht mehr der typischen Vorgehensweise:

1. Design der GUI(s)
2. Entwicklung der Datenstrukt(en)
3. Deklaration der globalen Daten
4. Initialisierung der Daten und Datenstrukturen
5. Programmieren der Callbacks
6. Programmieren der Anwendung außerhalb der Callbacks

Also ran ans Werk. Die erste Callback-Routine, über die jede Anwendung schon bereits während ihrer Entwicklung verfügen sollte, ist der Callback zum Beenden der Anwendung. Also füttern wir in `win1.icn` die Prozedur `Win1_quit()`.

/home/pi/icon9_51/bin/win1.icn

```
1 procedure Win1_quit(vidget, value)
2   exit()
3 end
```

Das Gleiche machen wir dann auch gleich für win2.icn, compilieren `MehrereFenster.icn` und klicken mal auf die Schaltfläche `Quit`. Gut, was? Sowohl Anklicken der Schaltfläche `Quit` von `Win1` also auch der von `Win2` beendet die im Entstehen begriffene Anwendung.

Diese Methodik ist genau dann besonders vorteilhaft einzusetzen, wenn außer den Bedienelementen der einzelnen Fenster auf keine weiteren Ereignisse zu reagieren ist. Also keine Mausereignisse, keine Tastaturereignisse, ...
Wer dagegen auch solche Ereignisse betrachten möchte, der muss eine ganz klassische Ereignisbehandlung programmieren.

25.3.8.6 Annäherung 6: Klassische Ereignisbehandlung

Die klassische Ereignisbehandlung in Icon besteht darin, dass als erstes geprüft wird,

- ob überhaupt auf irgendeinem Fenster ein Ereignis vorhanden ist (Zeile 36)
- Wenn ein Ereignis vorliegt, wird geprüft, welches Fenster ein schwebendes Ereignis gemeldet hat (Zeile 37)
- Dann wird das Ereignis pauschal der bekannten Prozedur `ProcessEvent()` zugestellt – allerdings benutzen wir hier `ProcessEvent()` in der maximal möglichen Ausstattung
  `Ereignis := ProcessEvent(Root-Vidget des Fensters, QuitCheck, unbedingt, resize)`
  `ProcessEvent()` liefert ein Ereignis zurück, wenn dieses von `ProcessEvent()` nicht hatte verarbeitet werden können. Dies ist dann der Fall, wenn kein Vidget-Ereignis vorliegt, sondern ein Tastatur- oder Maus-Ereignis. Aus diesem Grund ist es wichtig, dieses Ereignis zu erhalten, um es in der weiteren Ereignisbehandlung verwenden zu können. `QuitCheck()` ist eine Prozedur, die aufgerufen wird, um Tastatur-Ereignisse, die zum Verlassen des Programms verwendet werden, zu entdecken und das Programm zu verlassen
- `unbedingt()` ist eine Prozedur, die unabhängig von irgendwelchen Bedingungen auf jeden Fall durchgeführt werden soll (Zeilen 11 bis 16). Hier bietet es sich während der Programmentwicklung beispielsweise an, Ereignisse und Mauskoordinaten etc. anzeigen zu lassen.
- `resize()` ist eine Prozedur, die aufgerufen werden soll, wenn das Programm ein `Resize`-Ereignis empfängt. Um das zu testen, habe ich die Fensterattribute von `Win1` entsprechend durch `"resize=on"` ergänzt (Zeile 67).
- Anschließend werden alle Tastatur- (Zeile 39 bis 40) und Mausereignisse (Zeile 41) geprüft und bei Übereinstimmung entsprechende Prozeduren, die auch die klassischen Callback-Prozeduren sein können, aufgerufen – oder eigene Ausdrücke in beliebiger Komplexität eingesetzt.
- Während der Entwicklung sollte die Fallunterscheidung wie im Beispiel angegeben `default`-Fälle abfragen, da hier am einfachsten erkannt werden kann, ob Ereignisse unberücksichtigt geblieben sind (Zeilen 42 bis 44).

- Zum Schluss sollte sich der Programmierer immer vergewissern, dass die Ereignisbehandlung sich so verhält, dass seine Anwendung nicht die einzige im System ist, die Prozessor-Zeit erhält. Ein simples `delay()` oder `WDelay()` sorgt dafür, dass die CPU-Usage zumindest auf dem Raspberry Pi von 100 % auf weniger als 5% sinken kann, indem eine sinnvolle Verzögerung definiert wird.

25.3.8.7 Annäherung 7: Zyklus Optimierung der Ereignisbehandlung, der Callback-Routinen und der graphischen Benutzeroberflächen

In einem solchen Zyklus werden weitere Features eingebaut. In unserem Beispiel soll bei Drücken der rechten Maustaste in dem einen Fenster die Zeichenfarbe im anderen Fenster zufällig gesetzt werden. Dies wird in `MehrereFenster.icn` in Zeile 44 innerhalb der Ereignisbehandlung geprüft und umgesetzt. Dies ist ein Beispiel, in der eine Ereignisbehandlung nicht in eine Callback-Routine endet. Bei übersichtlichen Ausdrücken empfiehlt sich diese Vorgehendweise anstelle einer eigenen Prozedur. Das ist aber Ansichtssache – und alle Möglichkeiten sind erlaubt.
In diesen Zyklus fällt auch, eine globale Liste zu deklarieren (Zeile 10), in die die Uhrzeiten eingetragen werden, sobald die jeweilige `Time`-Schaltfläche angeklickt wird (`win1.icn`, `win2.icn`: Zeilen 55 bis 58).
Meistens steigen die Vidgets-Variablen und die Root-Variablen von lokalen Variablen der ehemaligen Prozedur `main()` der Programme `win1.icn` und `win2.icn` zu globalen Variablen in `MehrereFenster.icn` auf (Zeile 9), da diese Variablen überall benötigt werden.

Schließlich werden innerhalb dieses Zyklus dann auch die Callback-Prozeduren `Win1_time()` und `Win2_time()` mit Leben gefüllt (dort in den Zeilen 55 bis 58). Abschließend habe ich mir noch überlegt, dass es ganz praktisch ist, bei Anklicken auf das Listenfeld die Liste zu löschen (Zeilen 60 bis 63).

25.3.8.8 Annäherung 8: Löschen aller nicht mehr benötigten Zeilen, Hilfsausgaben etc.

Während der meisten Anwendungsentwicklungen lässt sich der Programmierer für Kontrollzwecke hier und da Werte von Variablen, Datenstrukturen und Datensätzen ausgeben, um beurteilen zu können, ob das Programm mit den Daten auch die gewünschten Operationen durchführt. Diese Zeilen werden – sobald die Prozedur bzw. das Modul wunschgemäß funktioniert – im Verlauf der Anwendungsentwicklung nach und nach entfernt.

Ich selber habe mir angewöhnt, dieses Löschen in mehreren Stufen vorzunehmen. Zunächst werden die betreffenden Anweisungen in Kommentare gesetzt – man kann ja nie wissen, ob eine spätere Kontrolle doch wieder erforderlich werden könnte. Sobald aber definitiv feststeht, dass der betreffende Programmteil exakt funktioniert (und ggf. auch auf extreme Daten im Rahmen sog. Stress-Tests richtig reagiert), dann werden diese Zeilen endgültig gelöscht.

Um diese Zeilen leichter identifizieren zu können, setze ich solche – später auf jeden Fall zu löschenden Zeilen – in die erste Spalte des Programmcodes. Da diese Kontrollausgaben auf Höhe von `procedure` und `end` stehen, fallen diese auf jeden Fall immer auf.

/home/pi/icon9_51/bin/MehrereFenster.icn

```
1  link graphics
2
3  #$include "keysyms.icn"# Falls auf Tastatur-Ereignisse reagiert werden soll: Kommentarzeichen
4  entfernen
5  $include "win1.icn"
6  $include "win2.icn"
7
   global Win1_vidgets, Win1_root,
       Win2_vidgets, Win2_root,
8      Ereignis,
9      Win1_liste, Win2_liste
10
11 procedure unbedingt()
12   if /Ereignis then return
   # case type(Ereignis) of
   # { "integer": #Notice(Ereignis)
13 #   "string":  #Notice(Ereignis)
```

```
14  #    default: #Notice(type(Ereignis))
15  # }
16  end
17
18  #procedure resize()
19  #
20  #end
21
22  procedure main()
23    Win1_liste := []
24    Win2_liste := []
25
26    Win1  :=  WOpen  !  Win1_ui_atts()  |  stop("Fenster  Win1  kann  nicht  geöffnet  werden!  -
    Programmabbruch")
27    Win1_vidgets := Win1_ui()
28    Win1_root := Win1_vidgets["root"]
29    &window := &null
30
31    Win2  :=  WOpen  !  Win2_ui_atts()  |  stop("Fenster  Win2  kann  nicht  geöffnet  werden!  -
    Programmabbruch")
32    Win2_vidgets := Win2_ui()
33    Win2_root := Win2_vidgets["root"]
34
35    repeat
36    { while *Pending(Win1 | Win2) > 0 do
37      { case win := Active() of
38        { Win1: { case Ereignis := ProcessEvent(Win1_root, QuitCheck, unbedingt, resize) of
39                  { #"q" | "Q":     {Win1_quit()}
40                    "t" | "T":      {Win1_time()}
41                    &rrelease:      {Fg(Win2, ?65535,?65535,?65535)}
42                    #default: {#ProcessEvent(Win1_root)
43                    #          Notice(win, "Unbekantes Ereignis ", image(Ereignis), " in Fenster
    ", image(win))
44                    #         }
45                }
46            }
47          Win2: { case Ereignis := ProcessEvent(      Win2_root, QuitCheck,
48                                                      unbedingt,
49                                                      resize
50                                                ) of
51                { #"q" | "Q":     {Win2_quit()}
52                  "t" | "T":      {Win2_time()}
53                  &rrelease:      {Fg(Win1, ?65535,?65535,?65535)}
54                  #default: {#ProcessEvent(Win2_root)
55                  #          Notice(win, "Unbekantes Ereignis ", image(Ereignis), " in Fenster
    ", image(win))
56                  #         }
57                }
58            }
59          #default:  { Notice("Ereignis in unbekantem Fenster ", image(win))
60          #      }
61        }
62      }
63      WDelay(50)
64    }
65
66    WDone()
67  end
```

/home/pi/icon9_51/bin/win1.icn

```
1   ############################################################################
2   #
3   # File:     /home/andreas/icon9_51/bin/win1.icn
4   #
5   # Subject:  Program to ...
6   #
7   # Author:
8   #
9   # Date:     January 14, 2015
10  #
11  ############################################################################
12  #
13  #
14  #
15  ############################################################################
16  #
17  #  Requires:
18  #
19  ############################################################################
20  #
21  #  Links: vsetup
```

```
#
############################################################################

#  This vib interface specification is a working program that responds
#  to vidget events by printing messages.  Use a text editor to replace
#  this skeletal program with your own code.  Retain the vib section at
#  the end and use vib to make any changes to the interface.

link vsetup

procedure Win1_quit(vidget, value)
  exit()
end

procedure Win1_time(vidget, value)
   put(Win2_liste, &clock)
   VSetItems(Win2_vidgets["Win2_list"], Win2_liste)
end

procedure Win1_list(vidget, value)
  Win1_liste := []
  VSetItems(Win1_vidgets["Win1_list"], Win1_liste)
end

#===<<vib:begin>>=== modify using vib; do not remove this marker line
procedure Win1_ui_atts()
   return ["size=125,400", "bg=pale gray", "label=Win1", "resize=on"]
end

procedure Win1_ui(win, cbk)
return vsetup(win, cbk,
   [":Sizer:::0,0,125,400:Win1",],
   ["Win1_list:List:w::12,96,97,282:",Win1_list],
   ["Win1_quit:Button:regular::11,9,100,40:QUIT",Win1_quit],
   ["Win1_time:Button:regular::11,51,100,40:Time to Win2",Win1_time],
   )
end
#===<<vib:end>>=== end of section maintained by vib
```

/home/pi/icon9_51/bin/win2.icn

```
############################################################################
#
# File:     /home/andreas/icon9_51/bin/Win2.icn
#
# Subject:  Program to ...
#
# Author:
#
# Date:     January 14, 2015
#
############################################################################
#
#
#
############################################################################
#
#  Requires:
#
############################################################################
#
#  Links: vsetup
#
############################################################################

#  This vib interface specification is a working program that responds
#  to vidget events by printing messages.  Use a text editor to replace
#  this skeletal program with your own code.  Retain the vib section at
#  the end and use vib to make any changes to the interface.

link vsetup

procedure Win2_quit(vidget, value)
  exit()
end

procedure Win2_time(vidget, value)
   put(Win1_liste, &clock)
   VSetItems(Win1_vidgets["Win1_list"], Win1_liste)
end

procedure Win2_list(vidget, value)
```

```
42    Win2_liste := []
43    VSetItems(Win2_vidgets["Win2_list"], Win2_liste)
44 end
45
46 #===<<vib:begin>>=== modify using vib; do not remove this marker line
47 procedure Win2_ui_atts()
48    return ["size=125,400", "bg=pale gray", "label=Win2"]
49 end
50
51 procedure Win2_ui(win, cbk)
52 return vsetup(win, cbk,
53    [":Sizer:::0,0,125,400:Win2",],
54    ["Win2_list:List:w::12,96,97,282:",Win2_list],
55    ["Win2_quit:Button:regular::11,9,100,40:QUIT",Win2_quit],
56    ["Win2_time:Button:regular::11,51,100,40:Time to Win1",Win2_time],
57    )
58 end
59 #===<<vib:end>>=== end of section maintained by vib
```

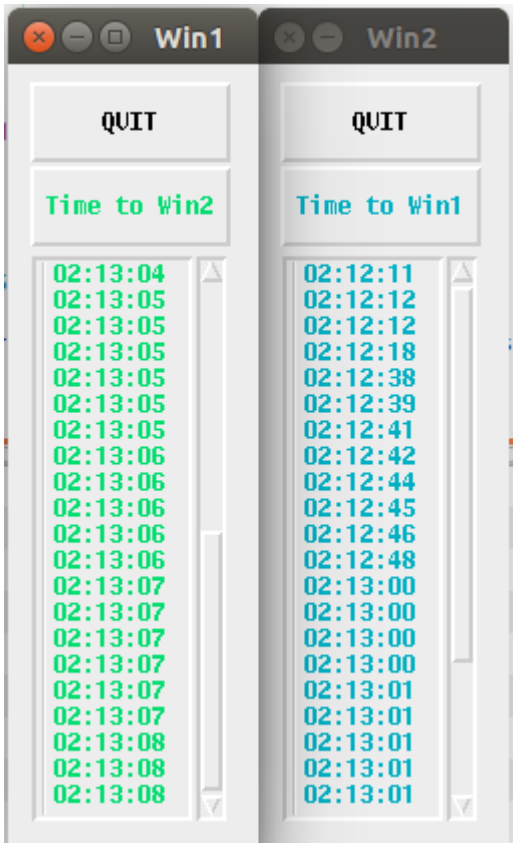

Abbildung 25-12: Screenshot der fertigen Anwendung in Aktion

Übungsaufgabe:

Schreibe eine Anwendung, deren sechs gleichgroße Fenster den gesamten Bildschirm ausfüllen. Jedes Fenster soll auf Tastenereignisse reagieren. Alle Tastenereignisse, die ein sichtbares Zeichen erzeugen, sollen auf bestimmten Fenstern ausgegeben werden:

- Fenster 1: Nimmt alle Vokale auf
- Fenster 2: Nimmt alle Konsonanten auf
- Fenster 3: Nimmt alle Kleinbuchstaben auf
- Fenster 4: Nimmt alle Großbuchstaben auf
- Fenster 5: Nimmt alle Ziffern auf
- Fenster 6: Nimmt alle anderen sichtbaren Zeichen auf

Der Fokus wird durch Anklicken der jeweiligen Fensterfläche verschoben sowie durch die Funktionstasten F1 bis F6 und Schaltflächen ermöglicht.

Jeder Fokuswechsel soll in einer Logdatei mit der Uhrzeit, zu der der Fokus des Fensters erhalten wurde, aufgezeichnet werden. Ein Eintrag soll wie folgt aussehen:

```
dd.mm.yyyy, hh:mm:ss: Fenster x ==> Fenster y: Liste der Ereignisse
```

Alle nicht-sichtbaren Zeichen (z.B. Tab-Taste, Enter-Taste) wirken sich direkt auf das Fenster aus, dass dieses Ereignis erhalten hat. Ich denke, man muss nicht extra erwähnen, dass

- jedes Fenster seinen eigenen Titel erhält,
- jedes Fenster über Schaltflächen zum Verlassen des Fensters verfügt
- jedes Fenster ein Menü besitzt, über das z.B. die Hintergrund- und Vordergrundfarben des jeweiligen Fensters eingestellt werden kann. Um es ein wenig zu vereinfachen, nehmen wir nur die von Icon zur Verfügung gestellten Grundfarben zur Auswahl.

Deine Funktionstasten zum Lauter und Leiser stellen sollen dafür verwendet werden können, ab diesem Zeitpunkt die Schriftzeichen größer bzw. kleiner darzustellen.

Was kommt demnächst?
26 Individuelle Dialoge mit VIB erstellen