




**FOX** embedded computers   
*the canny swiss solution*

**Workshop „Touchpanel-Anwendungen mit dem eigerPanel 57“  
Unterägeri, 4.11.2008**

# Step by Step ... zum Kitchen Timer

## Step 1 Idee

Kitchen-Timer mit:

- Rechtecke
- Symbole
- Labels
- Buttons (animiert, windows behaviour)
- Berührungsempfindliche Zonen
- Uhr / Timer
- Alarm mit eingebautem Buzzer
- Hintergrundbild



Abbildung 1: So wird der Kitchen Timer aussehen



## Step 2 Layout-Entwurf

Layout in Excel vorbereiten

- Hilfsgitter (Gitterabstand 10 Pixel → 64 x 48 Felder = VGA)
- Rahmen einfärben
- Buttons dimensionieren und positionieren

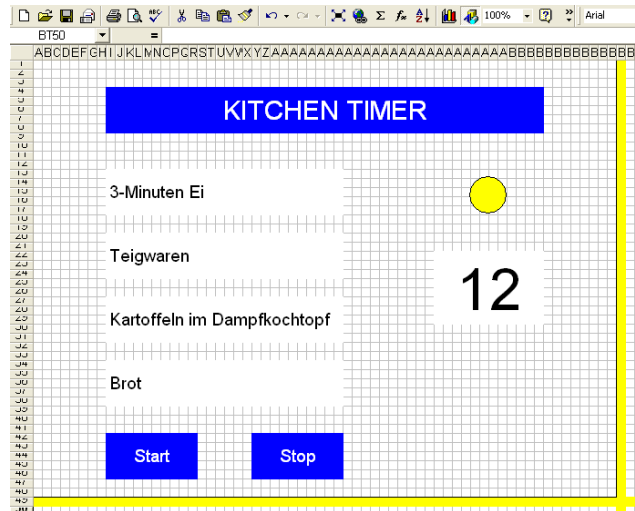


Abbildung 2: Entwurf des Layouts z.B. mit Hilfe von Excel . Ein Feld entspricht 10 x 10 Pixel.

## Step 3 Vorbereitungen für Eigerprojekt

### 3.1 Dateistruktur und Headerdateien

- Projektverzeichnis → 4 Zeichen: KTIM
- Unterverzeichnisse (individuell): PICT, EIGER, DUMP, DATA, HOST
- Headerdateien ins Verzeichnis EIGER

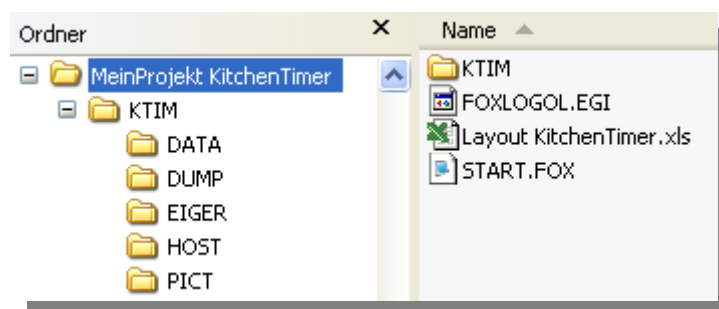


Abbildung 3: Verzeichnisstruktur des eigerProjektes KTIM

### 3.2 Startbild und Startdatei

- Startbild → FOXLOGOL.EGI bzw. ...P.EGI
- Startdatei → START.FOX (wichtig: Syntax genau beachten: Semikolon am Schluss gehört dazu)

### 3.3 EVS-Datei eröffnen

- EigerStudio öffnen (2x, d.h. einmal leer und einmal mit Vorbildprojekt KTIM)
- Datei als KTIM.EVS ins Verzeichnis KTIM speichern
- Infokopf ins EVS kopieren
- Obligatorischer Bezeichnungs-Code:

#### Code 1

```
EIGERPROJECT 'KTIM' ; Projektbezeichnung: erster Teil des
    EVI-Filenamens
EIGERVIEW 001 ; Viewnummer: zweiter Teil des EVI-
    Filenames: KTIM_001.EVI
```

Code-Erklärung:

Aus diesen beiden Bezeichnungen **KTIM** und **001** setzt sich der Dateiname der View zusammen: „KTIM\_001.EVS“. Der Dateiname besteht immer aus 8 Zeichen. Der Compiler bedient sich dieser beiden Definitionen, um die Datei „KTIM\_001.EVI“ zu generieren, die den Byte-Code für das eigerPanel 57 enthält.

- Datei als KTIM\_001.EVS speichern ins Verzeichnis KTIM
- Links zu Standard Include-Dateien einfügen (mit relativem Pfad)

#### Code 2

```
; Standard Eiger Includes -----
IMPORT 'EIGER/DEF_eVM_OpCodes_0$70.h' ; Tokens
IMPORT 'EIGER/DEF_eVM_Registers_0$70.h' ; Register
FUNCLIB 'EIGER/DEF_eVM_Functions_0$70.lib' ; Funktionsbibliothek
INCLUDEFILE 'EIGER/DEF_eiger_Colors_0$70.INC' ; Farbdefinitionen
INCLUDEFILE 'EIGER/DEF_eiger_Types_0$70.INC' ; Eiger Definitionen
```

## Step 4 Definitionen

### 4.1 Farbdefinitionen

- eigerStudio hat vorgegebene Farbkonstanten → gespeichert in DEF\_eiger\_Colors\_0\$70.INC
- Farben können auch selbst festgelegt werden durch Definieren der RGB - Anteile:

#### Code 3

```
Colors.Load_RGB(VarInt:Ziel,VarInt:R,VarInt:G,VarInt:B)
```

- Definitionen für benutzte Farben (wir benutzen vorgegebene Farbkonstanten)

#### Code 4

```
; Farbdefinitionen für das Layout -----
CONST TitleColor = blue181
CONST SelectionColor = beige
CONST ButtonColor = blue181
CONST TimerColor = beige
```



## 4.2 Definition der Objekt-Koordinaten und -Dimensionen

- Objekte sind in unserem Beispiel Labels, Buttons und Rahmen für Timer.
- Koordinaten, Breiten und Höhen werden als **Anzahl Pixel** angegeben.

### Code 5

```

; Koordinaten für das Layout -----
CONST LayoutLeft      = 80           ; linker Rand
CONST LayoutTop       = 35           ; oberer Rand

CONST TitleLeft       = LayoutLeft   ; linker Rand des Titels
CONST TitleTop        = LayoutTop    ; linker Rand des Titels
CONST TitleHeight     = 50           ; Höhe des Titels
CONST TitleWidth      = 480          ; Breite des Titels

CONST ButtonLeft      = LayoutLeft   ; linker Rand des Start-Buttons
CONST ButtonWidth     = 100          ; Breite des Start- und Stop-Buttons
CONST ButtonHeight    = 50           ; Höhe des Start- und Stop-Buttons
CONST ButtonSpace_X   = 70           ; X-Abstand des Start- und Stop-Buttons

CONST SelectionLeft   = LayoutLeft   ; linker Rand der Vorwahl-Buttons
CONST SelectionTop    = TitleTop + TitleHeight + 30 ; oberer Rand des ersten
    Vorwahl-Buttons
CONST SelectionWidth  = (2*ButtonWidth) + ButtonSpace_X ; Breite der Vorwahl-
    Buttons
CONST SelectionHeight = 50           ; Höhe der Vorwahl-Buttons
CONST SelectionSpace_Y = 20          ; Y-Abstand der Vorwahl-Buttons

CONST TimerLeft       = 440          ; linker Rand des Timers
CONST TimerTop        = 200          ; oberer Rand des Timers
CONST TimerWidth      = 120          ; Breite des Timers
CONST TimerHeight     = 80           ; Höhe des Timers

; oberer Rand des Start- und Stop-Buttons in Abhängigkeit der Vorwahlen
CONST ButtonTop = SelectionTop + (4*SelectionHeight) + (4*SelectionSpace_Y) + 5

```

## Step 5 Hauptprogramm

- Eine klare Strukturierung verkürzt das Suchen (z.B. 1. Deklarationen, 2. Subroutinen, 3. Hauptprogramm, 4. Styles)
- Hauptprogramm steuert den Programmverlauf → Code wird der Reihe nach abgearbeitet
- Hauptprogramm ist umrahmt mit den Schlüsselwörtern `BEGINVIEW` und `ENDVIEW`. `eigerStudio` erkennt die Schlüsselworte seiner Programmiersprache während der Eingabe und formatiert diese automatisch. Weitere Schlüsselworte sind beispielsweise `EIGERPROJECT`, `EIGERVIEW`, `CONST`, `STRING`, `INTEGER`, `SUB` und `ENDSUB`.
- Code für Beginn des Hauptprogramms:



**Code 6**

```
BEGINVIEW
    ; Grafikcontroller initialisieren -----
    EVE.Init()

    ; Hintergrundzeichenen einschalten -----
    Display.Prepare()
```

**Code-Erklärung:**

Mit `EVE.Init()` wird die Grafikmaschine EVE (eigerVideoEngine) frisch initialisiert. Damit wird EVE von älteren Einstellungen und noch anstehenden Events (z.B. Bildschirmberührungen) gereinigt. So werden unerwünschte Überraschungen vermieden.

`Display.Prepare()` sorgt für einen Viewaufbau im Hintergrund, d.h. im „hinteren“ Videospeicher (Refresh Video RAM **RVR**), der nicht direkt mit dem Display verbunden wird. Mit `Display.Show()` kann dann das zuvor schrittweise aufgebaute Layout in *einem* Augenblick dargestellt werden → es wird in den „vorderen“ Videospeicher (Accessible Video RAM **AVR**) kopiert, dessen Inhalt direkt im Display angezeigt wird (vgl. eigerScript – Schnelleinstieg, S.46-48).

**Step 6 Speichern und Kompilieren**

- Sobald die View ein Hauptprogramm enthält, ist sie kompilierbar.
- Kompilieren mit Button in der Symbolleiste, mit F5 oder mit *Compiler > Compile Source*.
- Bei fehlerhaftem Code kann das Programm nicht kompiliert werden. In diesem Fall hilft die Fehlerbeschreibung im Log Window (unteres Teilfenster) von eigerStudio weiter.

**Step 7 Zeichnen von Rechtecken, Kreisen und Linien****7.1 Anlegen einer Subroutine**

- Eine Subroutine ist gekennzeichnet durch die Schlüsselwörter `SUB` und `ENDSUB`.
- Wir nennen die Subroutine „Rechteck“ → `SUB Tree`

**Code 7**

```
SUB Tree
ENDSUB
```

- Aufruf aus dem Hauptprogramm; die Subroutine wird nur aktiv, wenn sie vom Hauptprogramm aus aufgerufen wird:

**Code 8**

```
CallSubroutine(Tree)
```



- Das Rechteck soll nicht nur im Hintergrund vorbereitet, sondern auch auf dem Display dargestellt werden, deshalb braucht es am Ende des Hauptprogramms noch den Befehl:

**Code 9**

```
Display.Show()
```

## 7.2 Code für Rechteck

- Löschen des Startbildes, d.h. mit weiss übermalen, durch Befehl zu Beginn des Hauptprogramms

**Code 10**

```
BEGINVIEW  
  
; Grafikcontroller initialisieren -----  
EVE.Init()  
  
; "leeren" Bildschirm zeichnen -----  
Display.ClearColor( white )
```

1. Laden der Rechteck-Koordinaten mit X (→) und Y (↓) für linken oberen Eckpunkt.
2. Laden von Breite (W) und Höhe (H) des Rechtecks.
3. Festlegen von Farbe und Rahmen mit den entsprechenden Registern.
4. Befehl zum Zeichnen des Rechtecks; ausgefüllt mit `Draw.RectangleFilled()` und nur Rahmen mit `Draw.Rectangle()`.

→ vgl. Code 11 und Abbildung 4

## 7.3 Code für Kreis

- Festlegen der Koordinaten des Zentrums
- Festlegen des Radius, z.B. 40 Pixel
- Festlegen der Füllfarbe (z.B. mit der Farbkonstante „darkgreen“) und der Rahmenfarbe (wir lassen sie braun)

→ vgl. Code 11 und Abbildung 4

## 7.4 Code für Linie

- Für das Zeichnen einer Linie gibt es diverse Methoden. Wir zeichnen eine Linie als Verbindung von Punkt 1 (X1,Y1) und Punkt 2 (X2,Y2).
- Festlegen des Anfangspunktes (identisch mit Kreismittelpunkt) und des Endpunktes (X2 = 250, Y2 = 290)

→ vgl. Code 11 und Abbildung 4



## Code 11

```
SUB Tree

; Rechtecke
Load.Geometry_XYWH( 200, 300, 100, 50 )
eI.FillColor := yellowgreen
eI.LineColor := brown
Draw.RectangleFilled() ; ausgefülltes Rechteck

Load.Geometry_XYWH( 210, 290, 80, 10 )
Draw.Rectangle() ; nur Rechteck-Rahmen

; Kreis
Load.Pos_X1Y1( 250, 200 )
eI.Radius := 40
eI.FillColor := darkgreen
Draw.CircleFilled

; Linie
Load.Pos_X2Y2( 250, 290 )
Draw.Line()

ENDSUB
```

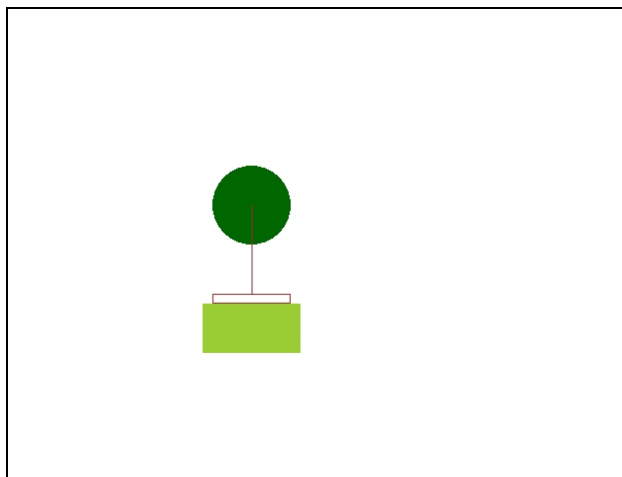


Abbildung 4: Baum aus Rechtecken, Kreis und Linie als Ergebnis der Subroutine SUB Tree. (ScreenShot)

## 7.5 Löschen oder auskommentieren

Dieser Baum hat jedoch nichts mit dem Kitchen Timer zu tun, darum löschen wir die Subroutine SUB Tree sowie den entsprechenden Verweis im Hauptprogramm oder wir kommentieren es aus:

- Auskommentieren einer Zeile: Semikolon am Zeilenanfang → die Zeichen färben sich grün
- Auskommentieren mehrerer markierter Zeilen: *Edit > Comment* → am linken Rand der markierten Zeilen wird ein Semikolon gesetzt.

## Step 8 Titel-Label für den Kitchen-Timer

### 8.1 Style für Titel-Label



- Die Grafikparameter für ein Label werden häufig „en bloc“ festgelegt, d.h. mit einem **Style**.
- Ein Style ist als ein in sich geschlossener Körper zu behandeln. Reihenfolge und Anzahl seiner Glieder dürfen nicht verändert werden!
- Mit dem Style werden die Register (z.B. `eI.Pos_X1`) bedient, die das Layout eines Labels bestimmen können.
- Die Styles sind in der Subroutine `SUB Styles` abgelegt.
- Den `Title_Style` „füllen“ wir mit den gewünschten Parametern und Konstanten, die wir am Anfang der View bereits deklariert haben.

#### Code 12

```

SUB   Styles
Title_Style:
  INLINERWORDS   (TitleLeft)           ; entspricht eI.Pos_X1
  INLINERWORDS   (TitleTop)            ; entspricht eI.Pos_Y1
  INLINERWORDS   (TitleWidth)          ; entspricht eI.Width
  INLINERWORDS   (TitleHeight)         ; entspricht eI.Height
  INLINERWORDS   (5)                   ; entspricht eI.SpaceLeft
  INLINERWORDS   (5)                   ; entspricht eI.SpaceRight
  INLINERWORDS   (0)                   ; entspricht eI.HorizontalAdjust
  INLINERWORDS   (0)                   ; entspricht eI.VericalAdjust
  INLINERWORDS   (TitleColor)          ; entspricht eI.FillColor
  INLINERWORDS   (as_FillColor)        ; entspricht eI.BackgroundColor
  INLINERWORDS   (as_FillColor)        ; entspricht eI.LineColor
  INLINERWORDS   (autocolor)           ; entspricht eI.TextColor
  INLINERWORDS   (Pos_center)          ; entspricht eI.Position
  INLINERWORDS   (Orientation_0deg)    ; entspricht eI.Orientation
  INLINERWORDS   (normal)              ; entspricht eI.Appearance
  INLINERWORDS   (no_border)           ; entspricht eI.BorderStyle
  INLINERWORDS   (Font_Arial_24n)     ; entspricht eI.FontNumber
  INLINERWORDS   (silver)              ; entspricht eI.BackgroundColor
ENDSUB

```

### 8.2 Titel-Label zeichnen

- Den Titel-Code schreiben wir ins Hauptprogramm
- Style für das Titel-Label aktivieren mit `Fill.LabelParameter()`.
- Den Text für das Label festlegen entweder direkt in einfachen Anführungszeichen oder indirekt durch eine Stringvariable, die zuvor definiert worden ist.

#### Code 13

```

; Titel Zeichnen -----
Fill.LabelParameter( Title_Style )
Label.Text( 'KITCHEN TIMER' )

```

## Step 9 Vorwahlknopf 1 zeichnen

- Neue Subroutine für den ersten Vorwahlknopf → `SUB Select1_Leave`
- Style festlegen → `Selection_Style`
- In der Subroutine für Vorwahlknopf 1 die Parameter des `Selection_Style` laden.
- Position festlegen mit X1 und Y1 (Im Hauptprogramm werden wir den Subroutinen-Aufruf direkt nach dem Titel-Code platzieren. Deshalb enthält das Register `eI.Pos_X1` bereits die richtige Koordinate).
- Label-Text eingeben: '3-Minuten Ei'

### Code 14

```

; -----
;
;   HOTSPOTS EREIGNISROUTINEN
; -----

; Zeichnet den Vorwahlknopf '3-Minuten Ei' im inaktiven Zustand

SUB  Select1_Leave

    Fill.LabelParameter( Selection_Style )
    eI.Pos_Y1 := SelectionTop + 0 * ( SelectionHeight + SelectionSpace_Y )
    Label.Text( '3-Minuten Ei' )

ENDSUB

```

- Subroutinen-Aufruf ins Hauptprogramm schreiben

### Code 15: Subroutinen-Aufruf im Hauptprogramm

```

; Vorwahlen installieren -----
CallSubroutine( Select1_Leave )

```

## Step 10 HotSpot für Vorwahlknopf 1 installieren

Ein HotSpot ist eine fest definierte berührungsempfindliche Region auf dem Touchpanel. Schlussendlich soll beim Drücken auf den Vorwahlknopf 1 die entsprechende Kochzeit gewählt werden.

- Koordinaten und Dimensionen festlegen, z.B. mit `Load.Geometry_XYWH()`. Dies ist in unserem Fall nicht nötig, da wir den HotSpot direkt nach dem Subroutinen-Aufruf von Label 1 installieren (Code 16). Diese Reihenfolge garantiert uns, dass die Register noch mit den Koordinaten von Label 1 gefüllt sind. Dadurch erhält der HotSpot die gleiche Position und Grösse wie das Vorwahl-Label 1.
- HotSpot Installieren

### Code 16: Installieren des HotSpots im Hauptprogramm für Vorwahlknopf 1

```

; Vorwahlen installieren -----

```



```
CallSubroutine( Select1_Leave )
HotSpot.Install( NIL, NIL, Select1_Down, NIL )
```

Code-Erklärung:

Die Syntax eines HotSpots lautet: `HotSpot.Install( Enter, Leave, Down, Up )`. Demnach kann ein HotSpot zwischen vier Ereignissen unterscheiden:

Enter:	HotSpot wird erreicht (schieben des Fingers von aussen in den HotSpot)
Leave:	HotSpot wird verlassen (schieben des Fingers vom HotSpot nach aussen)
Down:	Berührung beginnt im HotSpot
Up:	Berührung endet im HotSpot

Unser erster HotSpot soll nur beim Ereignis „Down“ reagieren. Deshalb setzen wir die drei anderen Ereignisse mit `NIL` ausser Kraft. Sobald jemand auf den Vorwahlknopf 1 drückt, wird die Subroutine `SUB Select1_Down` aufgerufen.

- Mit `HotSpot.TableEnable` muss die HotSpotTable am Ende des Hauptprogramms freigegeben werden (Code 17). Erst mit diesem Befehl ist der HotSpot für Events, d.h. für einen Fingerdruck auf dem Touchpanel empfänglich.
- `LOOP` und `ENDLOOP` ganz am Ende des Hauptprogramms halten die Aufmerksamkeit der virtuellen Maschine EVE gegenüber eintreffenden Events aufrecht (Code 17).

Code 17

```
; Alle Hotspots und Timer aktivieren -----
HotSpot.TableEnable()

; Endlosschlaufe, von nun an läuft das Programm ereignisgesteuert -----
LOOP
ENDLOOP

ENDVIEW
```

## Step 11 Aktivierten Vorwahlknopf 1 zeichnen

Beim Drücken auf den Vorwahlknopf soll er sich rot verfärben und die Schrift weiss erscheinen.

- Neue Subroutine, die beim Down-Ereignis aufgerufen wird → `SUB Select1_Down`
- Style festlegen → `Selection_Style` (ist bereits vorhanden)
- In der Subroutine für Vorwahlknopf 1 die Parameter des `Selection_Style` laden.
- Label-Farbe wählen. Die Farbe des aktivierten Vorwahlknopfs unterscheidet sich von der Füllfarbe im `Selection_Style`. Diese muss nachträglich noch speziell dem Register `eI.FillColor` zugewiesen werden, z.B. mit der Methode `Label.Color( crimson )` oder mit `eI.FillColor := crimson`. Crimson ist die Farbkonstante für einen bestimmten Rot-Ton.
- Die Schriftfarbe ist im Style auf „autocolor“ gesetzt. Bei dunklen Farben wird die Schrift weiss.



## Code 18

```

; Wird einmal aufgerufen falls der Vorwahlknopf '3-Minuten Ei' gedrückt wird

SUB Select1_Down

    Fill.LabelParameter( Selection_Style )
    eI.Pos_Y1 := SelectionTop + 0 * ( SelectionHeight + SelectionSpace_Y )
    Label.Color( crimson )
    Label.Text( '3-Minuten Ei' )

; ; Alle anderen Vorwahlen deaktivieren
; CallSubroutine( Select2_Leave )
; CallSubroutine( Select3_Leave )
; CallSubroutine( Select4_Leave )
;
;
; ; Timer setzen
; TIMER.I := 4
;
; ; Vorwahl anzeigen
; CallSubroutine( TIMER_1_EXPIRED )

ENDSUB

```

## Code-Erklärung:

Die zweite Hälfte der Subroutine haben wir bisher noch nicht behandelt. Deshalb kommentieren wir diesen Teil vorerst noch aus.

## Step 12 Dito für Vorwahlknöpfe 2-4

Die weiteren drei Vorwahlknöpfe funktionieren nach dem gleichen Muster (vgl. Step 9 - Step 11). Wir können die Subroutinen `SUB Select1_Leave` und `SUB Select1_Down` einfach kopieren und dreimal untereinander anfügen, natürlich mit den entsprechenden Anpassungen:

## Anpassungen:

Vorwahlknopf 1	Vorwahlknopf 2	Vorwahlknopf 3	Vorwahlknopf 4
<code>SUB Select1_Leave</code>	<code>SUB Select2_Leave</code>	<code>SUB Select3_Leave</code>	<code>SUB Select4_Leave</code>
<code>eI.Pos_Y1 := SelectionTop + 0 * ( SelectionHeight + SelectionSpace_Y )</code>	<code>eI.Pos_Y1 := SelectionTop + 1 * ( SelectionHeight + SelectionSpace_Y )</code>	<code>eI.Pos_Y1 := SelectionTop + 2 * ( SelectionHeight + SelectionSpace_Y )</code>	<code>eI.Pos_Y1 := SelectionTop + 3 * ( SelectionHeight + SelectionSpace_Y )</code>
<code>Label.Text( '3-Minuten Ei' )</code>	<code>Label.Text( 'Kartoffeln im Dampfkochtopf' )</code>	<code>Label.Text( 'Teigwaren' )</code>	<code>Label.Text( 'Brot' )</code>
<code>SUB Select1_Leave</code>	<code>SUB Select2_Leave</code>	<code>SUB Select3_Leave</code>	<code>SUB Select4_Leave</code>
<code>eI.Pos_Y1 := SelectionTop + 0 * ( SelectionHeight + SelectionSpace_Y )</code>	<code>eI.Pos_Y1 := SelectionTop + 1 * ( SelectionHeight + SelectionSpace_Y )</code>	<code>eI.Pos_Y1 := SelectionTop + 2 * ( SelectionHeight + SelectionSpace_Y )</code>	<code>eI.Pos_Y1 := SelectionTop + 3 * ( SelectionHeight + SelectionSpace_Y )</code>
<code>Label.Text( '3-Minuten Ei' )</code>	<code>Label.Text( 'Kartoffeln im Dampfkochtopf' )</code>	<code>Label.Text( 'Teigwaren' )</code>	<code>Label.Text( 'Brot' )</code>

## Änderungen an den noch auskommentierten Zeilen:

<code>CallSubroutine( Select2_Leave )</code>	<code>CallSubroutine( Select1_Leave )</code>	<code>CallSubroutine( Select1_Leave )</code>	<code>CallSubroutine( Select1_Leave )</code>
<code>CallSubroutine( Select3_Leave )</code>	<code>CallSubroutine( Select3_Leave )</code>	<code>CallSubroutine( Select2_Leave )</code>	<code>CallSubroutine( Select2_Leave )</code>
<code>CallSubroutine( Select4_Leave )</code>	<code>CallSubroutine( Select4_Leave )</code>	<code>CallSubroutine( Select4_Leave )</code>	<code>CallSubroutine( Select3_Leave )</code>
<code>TIMER.I := 4</code>	<code>TIMER.I := 11</code>	<code>TIMER.I := 13</code>	<code>TIMER.I := 31</code>



In den Subroutinen `SUB Select#_Down` hatten wir die Zeilen bisher auskommentiert, welche die jeweils anderen Vorwahlen deaktivieren. Da nun alle Vorwahlknöpfe vorhanden sind, können wir diese grünen Zeilen nun markieren und mit `Edit > Uncomment` programmwirksam machen (vgl. Code 19). Dadurch kehrt nun beim wählen einer Kochzeit, der bisher aktivierte Vorwahlknopf in den deaktivierten Zustand zurück, während der neu gewählte sich rot färbt.

**Code 19:** „Kommentierte“ Zeilen werden durch Löschen der Semikolon zu „unkommentierten“ Programmzeilen

<pre>; Alle anderen Vorwahlen deaktivieren ;   CallSubroutine( Select2_Leave ) ;   CallSubroutine( Select3_Leave ) ;   CallSubroutine( Select4_Leave )</pre>	<pre>; Alle anderen Vorwahlen deaktivieren CallSubroutine( Select2_Leave ) CallSubroutine( Select3_Leave ) CallSubroutine( Select4_Leave )</pre>
--	--

Ebenso entfernen wir auch das Semikolon bei der Zeile, welche die Zeit für den Timer zuweist (Code 20). Dadurch wird beim Drücken des Vorwahlknopfs die Integer-Variable `Timer.I` mit der gewählten Kochzeit (Anzahl Minuten) gefüllt.

**Code 20:** „Kommentierte“ Zeilen werden durch Löschen der Semikolon zu „unkommentierten“ Programmzeilen

<pre>; Timer setzen ;   TIMER.I := 31</pre>	<pre>; Timer setzen TIMER.I := 31</pre>
---	---

Die Variable `Timer` haben wir noch nicht deklariert. Das holen wir noch nach anschliessend an die Konstanten-Deklarationen (Code 21).

**Code 21**

```
; Variablen -----
INTEGER      TIMER.I      ; der aktuelle Timer-Stand
```

Kompilieren wir das Projekt wieder einmal und überprüfen wir das Resultat auf dem eigerPanel 57...

Wir stellen fest, dass die drei neuen Vorwahlknöpfe (noch) nicht auf dem Display erscheinen. Wir haben vergessen, die Subroutinen der neuen Vorwahlknöpfe vom Hauptprogramm aus aufzurufen, wie wir das bereits beim ersten Vorwahlknopf getan haben (vgl. Code 16). Das holen wir nun nach, inkl. HotSpots (vgl. Code 22).

**Code 22:** Das Hauptprogramm wird mit den Verweisen zu den drei neuen Vorwahlknöpfen und deren HotSpots ergänzt

```
; Vorwahlen installieren -----
CallSubroutine( Select1_Leave )
HotSpot.Install( NIL, NIL, Select1_Down, NIL )

CallSubroutine( Select2_Leave )
HotSpot.Install( NIL, NIL, Select2_Down, NIL )

CallSubroutine( Select3_Leave )
HotSpot.Install( NIL, NIL, Select3_Down, NIL )

CallSubroutine( Select4_Leave )
HotSpot.Install( NIL, NIL, Select4_Down, NIL )
```

Nun erscheinen alle Vorwahlknöpfe auf dem Display.



## Step 13 Animierten Start-Button installieren

Den Start-Button positionieren wir entsprechend unserer Vorlage unterhalb des letzten Vorwahlknopfs.

### 13.1 Start-Button im Ruhezustand

Zunächst installieren wir den Start-Button in seinem Ruhezustand als „Button up“. Position, Dimension und die Layout-Parameter sind im `Button_Style_UP` definiert (Code 23).

Diesen Style kopieren wir aus dem Vorbildprojekt in die Subroutine `SUB Styles` unseres Projekts.

#### Code 23

```

Button_Style_UP:
  INLINEROWS      (ButtonLeft)           ; entspricht eI.Pos_X1
  INLINEROWS      (ButtonTop)            ; entspricht eI.Pos_Y1
  INLINEROWS      (ButtonWidth)          ; entspricht eI.Width
  INLINEROWS      (ButtonHeight)         ; entspricht eI.Height
  INLINEROWS      (4)                     ; entspricht eI.SpaceLeft
  INLINEROWS      (4)                     ; entspricht eI.SpaceRight
  INLINEROWS      (0)                     ; entspricht eI.HorizontalAdjust
  INLINEROWS      (0)                     ; entspricht eI.VericalAdjust
  INLINEROWS      (ButtonColor)           ; entspricht eI.FillColor
  INLINEROWS      (as_FillColor)          ; entspricht eI.BackColor
  INLINEROWS      (as_FillColor)          ; entspricht eI.LineColor
  INLINEROWS      (autocolor)             ; entspricht eI.TextColor
  INLINEROWS      (Pos_center)            ; entspricht eI.Position
  INLINEROWS      (Orientation_0deg)      ; entspricht eI.Orientation
  INLINEROWS      (normal)                ; entspricht eI.Appearance
  INLINEROWS      (color_button_3D_raised_big) ; entspricht eI.BorderStyle
  INLINEROWS      (Font_Arial_14n)        ; entspricht eI.FontNumber
  INLINEROWS      (silver)                ; entspricht eI.BackgroundColor

```

Die Subroutine `SUB Start_Leave` zeichnet den Start-Button im inaktiven Zustand (Code 24). Weshalb wir diese „`Start_Leave`“ und nicht „`Start_Up`“ nennen, sehen wir später.

#### Code 24

```

; Zeichnet den Button 'START' im inaktiven Zustand
SUB Start_Leave

  Fill.LabelParameter( Button_Style_UP )
  Label.Text( 'START' )

ENDSUB

```

Der Start-Button wird zu Programmstart gezeichnet, wenn im Hauptprogramm ein entsprechender Verweis eingetragen ist (Code 25).

#### Code 25

```

; Start-/Stop Buttons installieren -----
CallSubroutine( Start_Leave )

```



Wir prüfen unseren Fortschritt auf dem eigerPanel 57...

Dank eines speziellen Randes (Borders) erscheint der Start-Button dreidimensional. Im Style `Button_Style_UP` sehen wir, dass dieser Border „`color_button_3D_raised_big`“ heisst und im Register `eI.BorderStyle` abgelegt ist (vgl. Code 23, S.13). Die Farbe des Borders wird durch das Register `eI.LineColor` bestimmt.

eigerScript stellt noch viele weitere Borders zur Verfügung. Sie finden diese Border-Konstanten in der Headerdatei „`DEF_eiger_Types_0$50.INC`“.

### 13.2 Start-Button im „gedrückten“ Zustand

Wenn wir auf den Start-Button drücken, soll er dem Druck weichen, d.h. sich nach innen bewegen. Um ein diesen Eindruck visuell zu vermitteln, definieren wir den Style `Button_Style_DN` (Code 26). Der Border „`color_button_3D_sunk_big`“ vermittelt den „eingedrückten Eindruck“. Im Vorbildprojekt haben wir gesehen, dass der Button auf Druck nicht nur „eingedrückt“ erscheint, sondern gleichzeitig auch die Farbe auf orange wechselt. Im Down-Style sehen wir, dass den entsprechenden Registern die Farbkonstante „Dark Orange“ zugewiesen wird. Diesen Style kopieren wir vom Vorbildprojekt in die Style-Subroutine unseres Projekts. Der Button ändert auf Druck nicht nur

**Code 26:** Style für den eingedrückten Button in Darkorange mit dem Border `color_button_3D_sunk_big`

```

Button_Style_DN:
INLINENOWDS (ButtonLeft)           ; entspricht eI.Pos_X1
INLINENOWDS (ButtonTop)          ; entspricht eI.Pos_Y1
INLINENOWDS (ButtonWidth)        ; entspricht eI.Width
INLINENOWDS (ButtonHeight)       ; entspricht eI.Height
INLINENOWDS (4)                  ; entspricht eI.SpaceLeft
INLINENOWDS (4)                  ; entspricht eI.SpaceRight
INLINENOWDS (0)                  ; entspricht eI.HorizontalAdjust
INLINENOWDS (0)                  ; entspricht eI.VericalAdjust
INLINENOWDS (darkorange)         ; entspricht eI.FillColor
INLINENOWDS (as_FillColor)       ; entspricht eI.BackColor
INLINENOWDS (as_FillColor)       ; entspricht eI.LineColor
INLINENOWDS (autocolor)          ; entspricht eI.TextColor
INLINENOWDS (Pos_center)         ; entspricht eI.Position
INLINENOWDS (Orientation_0deg)   ; entspricht eI.Orientation
INLINENOWDS (normal)             ; entspricht eI.Appearance
INLINENOWDS (color_button_3D_sunk_big) ; entspricht eI.BorderStyle
INLINENOWDS (Font_Arial_14n)     ; entsprichetI.FontNumber
INLINENOWDS (silver)             ; entspricht eI.BackgroundColor

```

Den eingedrückten Button zeichnen wir mit Hilfe der Subroutine `SUB Start_Down` (Code 27). Diese weist den Grafik-Registern die Parameter des `Button_Style_DN` zu und gibt auch dem eingedrückten Button die Aufschrift „START“.

**Code 27**

```

; Wird einmal aufgerufen falls der Button 'START' gedrückt wird

SUB Start_Down

    Fill.LabelParameter( Button_Style_DN )
    Label.Text( 'START' )

ENDSUB

```



### 13.3 HotSpot für Start-Button im Hauptprogramm

Damit der Start-Button überhaupt reagiert, müssen wir den Bereich des Buttons berührungsempfindlich machen. Wir installieren dazu im Hauptprogramm einen HotSpot direkt nach dem Aufruf des Start-Buttons (Code 28). Dank dieser Positionierung ist die Angabe von Position und Dimension des HotSpots überflüssig, da die entsprechenden Register noch mit den richtigen Parametern gefüllt sind.

**Code 28:** HotSpot für Start-Button im Hauptprogramm direkt nach dem Aufruf zum Zeichnen des Start-Buttons

```
; Start-/Stop Buttons installieren -----
CallSubroutine( Start_Leave )
HotSpot.Install( NIL, Start_Leave, Start_Down, Start_Up )
```

Im neu installierten HotSpot sind die Events „Leave“ und „Down“ aktiviert. Sie weisen zu den Subroutinen `SUB Start_Leave` bzw. `SUB Start_Down`. Den Up-Event sollten wir vorläufig noch mit `NIL` belegen, da das Programm die Subroutine `Start_Up` noch nicht kennt (Code 29).

**Code 29:** Der Up-Event muss vorläufig noch inaktiv bleiben

```
HotSpot.Install( NIL, Start_Leave, Start_Down, NIL )
```

Wir überprüfen den neusten Projektstand auf dem eigerPanel 57...

- und drücken auf den Start-Button → er weicht und färbt sich rot.
- Wir drücken nochmals ohne loszulassen und schieben den Finger vom Button weg (= Leave Event) → der Button springt wieder zum Ruhezustand zurück.

## Step 14 Animierten Stop-Button installieren

Den Stop-Button installieren wir auf die gleiche Weise, wie den Start-Button. Den entsprechenden Code (Code 30) kopieren wir direkt unter den Code des Start-Buttons.

Auch dem Stop-Button weisen wir die Parameter des `Button_Style_UP` zu. Dabei müssen wir beachten, dass dieser Style die Position des Start-Buttons enthält. Wir müssen deshalb nach dem Laden des Styles den Positionsregistern noch speziell die X- und Y-Koordinaten des Start-Buttons zuweisen. Alle anderen Parameter sind mit dem Start-Button identisch.

**Code 30:** Code für den Stop-Button mit spezieller Zuweisung der X- und Y-Koordinaten

```
; -----
; Zeichnet den Button 'STOP' im inaktiven Zustand
```



```

SUB   Stop_Leave

    Fill.LabelParameter( Button_Style_UP )
    eI.Pos_X1 := ButtonLeft + ButtonWidth + ButtonSpace_X
    Label.Text( 'STOP' )

ENDSUB

; Wird einmal aufgerufen falls der Button 'STOP' gedrückt wird

SUB   Stop_Down

    Fill.LabelParameter( Button_Style_DN )
    eI.Pos_X1 := ButtonLeft + ButtonWidth + ButtonSpace_X
    Label.Text( 'STOP' )

ENDSUB

```

Auch hier vergessen wir die dazugehörigen Verweise und Einträge im Hauptprogramm nicht (Code 31).

**Code 31:** Verweis und HotSpot für Stop-Button, direkt nach dem Code für den Start-Button

```

; Start-/Stop Buttons installieren -----
CallSubroutine( Start_Leave )
HotSpot.Install( NIL, Start_Leave, Start_Down, NIL )

CallSubroutine( Stop_Leave )
HotSpot.Install( NIL, Stop_Leave, Stop_Down, NIL )

```

## Step 15 Timer

### 15.1 Style für Timer-Anzeige

Den Timer gestalten wir mit Hilfe eines separaten Styles (Code 32). Die Konstanten für Position, Dimension und Layout hatten wir bereits zu Beginn deklariert (vgl. Code 5).

**Code 32**

```

Timer_Style:
    INLINENWORDS (TimerLeft)           ; entspricht eI.Pos_X1
    INLINENWORDS (TimerTop)            ; entspricht eI.Pos_Y1
    INLINENWORDS (TimerWidth)          ; entspricht eI.Width
    INLINENWORDS (TimerHeight)         ; entspricht eI.Height
    INLINENWORDS (0)                   ; entspricht eI.SpaceLeft
    INLINENWORDS (0)                   ; entspricht eI.SpaceRight
    INLINENWORDS (0)                   ; entspricht eI.HorizontalAdjust
    INLINENWORDS (0)                   ; entspricht eI.VericalAdjust
    INLINENWORDS (TimerColor)          ; entspricht eI.FillColor
    INLINENWORDS (as_FillColor)         ; entspricht eI.BackColor
    INLINENWORDS (as_FillColor)         ; entspricht eI.LineColor
    INLINENWORDS (autocolor)           ; entspricht eI.TextColor
    INLINENWORDS (Pos_center)           ; entspricht eI.Position
    INLINENWORDS (Orientation_0deg)     ; entspricht eI.Orientation
    INLINENWORDS (normal)               ; entspricht eI.Appearance
    INLINENWORDS (color_button_3D_sunk_big) ; entspricht eI.BorderStyle
    INLINENWORDS (Font_DigitalNumbers_64) ; entspricht eI.FontNumber
    INLINENWORDS (silver)               ; entspricht eI.BackgroundColor

```



## 15.2 Timer-Anzeige zeichnen

Den Timer bringen wir mit den gleichen Befehlen auf den Bildschirm wie bereits die Vorwahlknöpfe und die Buttons. Die beiden Installations-Zeilen fügen wir direkt in das Hauptprogramm ein (Code 33).

**Code 33:** Installation des Timers im Hauptprogramm

```
; Timer-Anzeige installieren -----
Fill.LabelParameter( Timer_Style )
Label.Text( '00' )
```

## 15.3 Timer 1 im Hauptprogramm installieren

eigerScript stellt insgesamt acht Timer bereit, nummeriert von 0-7.

Ein Timer besteht aus drei Teilen:

- **Installation** des Timers im Hauptprogramm inkl. globaler Freigabe für alle Timer
- **Startbefehl** für Timer, meist in einer Subroutine
- **Anweisungen** in einer Subroutine, die vom Timer aufgerufen wird nach Ablauf der festgelegten Zeit.
- **Stopbefehl** für Timer.

Wir wählen den **Timer 1** und laden ihn mit **60000 ms** (= 1 Min). Wenn 1 Minute abgelaufen ist, soll er die zur Zeit noch unbekannt Subroutine `TIMER_1_EXPIRED` aufrufen. Dort wird dann geprüft, ob die gewählte Kochzeit nun abgelaufen ist, oder der Timer eine weitere Minute abwarten soll.

Mit diesen Vorgaben können wir den Timer im Hauptprogramm installieren (Code 34).

**Code 34**

```
; Timer für Kochzeit installieren -----
Timer.InstallLocal( 1, TIMER_1_EXPIRED )
Timer.Load( 1, 60000 ) ; 60000 msec = 1 Minute
```

Zusätzlich müssen wir am Ende des Hauptprogramms die TimerTable freigeben (Code 35).

**Code 35:** Am Ende des Hauptprogramms muss die TimerTable mit `Timer.TableEnable()` freigegeben werden.

```
; Alle Hotspots und Timer aktivieren -----
HotSpot.TableEnable()
Timer.TableEnable()

; Endlosschleufe, von nun an läuft das Programm ereignisgesteuert -----
LOOP
ENDLOOP

ENDVIEW
```



### 15.4 Subroutine für Timer-Start anlegen

Ein Timer kann als „Single Timer“ oder als „Continuous Timer“ konfiguriert werden. Näheres dazu finden Sie im eigerScript - Schnelleinstieg. Für unser Projekt eignet sich ein Continuous besonders gut, der jedes Mal, wenn die Zeit (1 Minute) abgelaufen ist, eine Aufgabe ausführt und den Countdown wieder von neuem startet. So können wieder alle Minuten prüfen lassen, ob nun die Kochzeit abgelaufen ist.

Der Timer soll beim Up-Event des Startbuttons gestartet werden, d.h. wenn wir den Start-Button nach dem Drücken wieder loslassen. Gleichzeitig soll der Start-Button wieder in seinen Ruhezustand mit Hilfe der Subroutine `SUB Start_Leave` (Code 24, S.13) zurückspringen. Für den eigentlichen Start des Timers sowie den Aufruf der `SUB Start_Leave` eröffnen wir eine eigene Subroutine `SUB Start_Up` (Code 36).

**Code 36:** Die Subroutine `start_Up` startet den Timer 1 als „Continuous Timer“ und veranlasst das Zeichnen des Start-Buttons im Ruhezustand durch Aufruf der Subroutine `start_Leave` (vgl. Code 24, S.13).

```

; Wird einmal aufgerufen falls der Button 'START' losgelassen wird

SUB  Start_Up

    ; Timer für Kochzeit starten
    Timer.StartContinuous( 1 )

    ; Wir zeichnen den Button wieder im inaktiven Zustand
    CallSubroutine( Start_Leave )

ENDSUB

```

Da nun auch die Aufgabe des Up-Events für den Start-Button formuliert ist, können wir beim HotSpot des Start-Buttons das „hinterste“ NIL mit dem Aufruf der Subroutine `Start_Up` ersetzen.

**Code 37:** Der Up-Event des Start-Buttons wird in der entsprechenden HotSpot-Methode mit dem Aufruf der Subroutine `start_Up` belegt (vgl. Code 36).

```

; Start-/Stop Buttons installieren -----
CallSubroutine( Start_Leave )
HotSpot.Install( NIL, Start_Leave, Start_Down, Start_Up )

```

### 15.5 Anweisungen bei Timer-Ablauf formulieren

Der Timer 1 soll den Countdown nur dann starten bzw. wiederholen, wenn der Wert in der Timer-Variablen `Timer.I` grösser als 0 ist. Diese Prüfung soll nach folgendem Schema ablaufen (Abbildung 5).

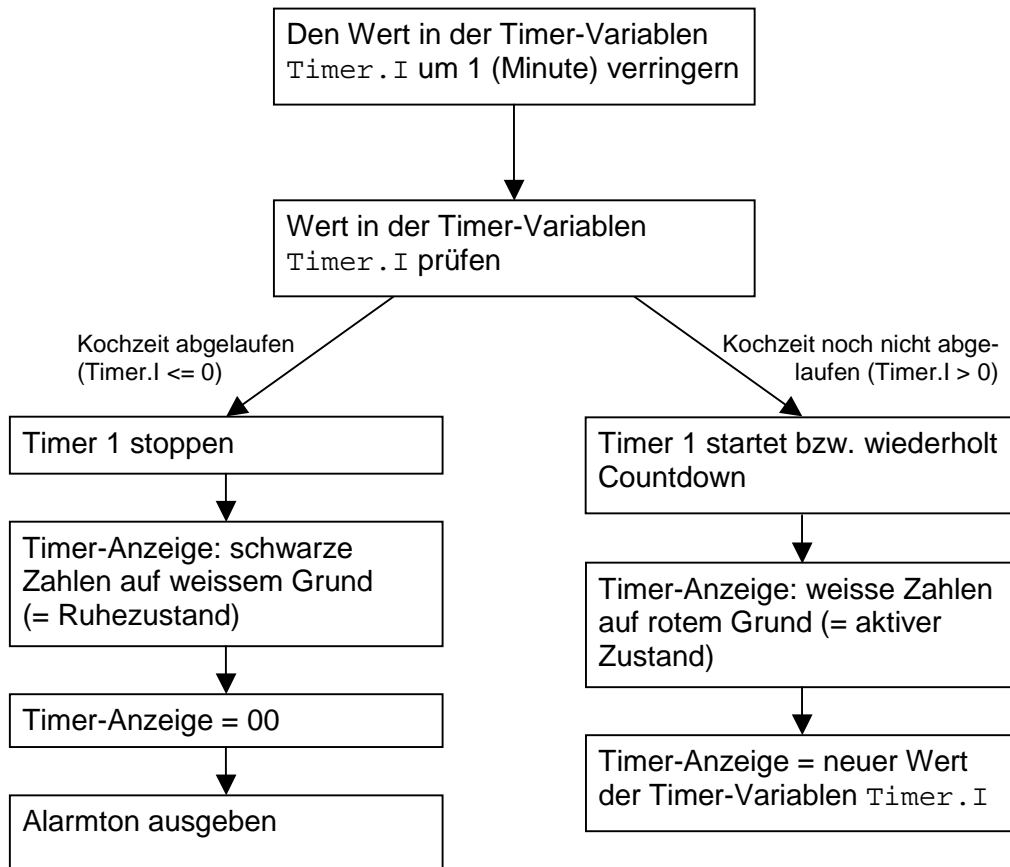


Abbildung 5: Prüfschema für den Continuous Timer 1

Die Prüfung, ob die Kochzeit abgelaufen ist, setzen wir in der `SUB TIMER_1_EXPIRED` um (Code 38).

**Code 38:** Prüft die verbleibende Kochzeit und zieht die entsprechenden Konsequenzen. Die Subroutine wird jedesmal aufgerufen, wenn ein Countdown des Continuous Timers 1 abgelaufen ist (vgl. Code 34).

```

; -----
;
;                               TIMER EREIGNISROUTINEN
; -----

; Kochzeit-Timer -----

SUB  TIMER_1_EXPIRED

; Style laden
Fill.LabelParameter( Timer_Style )

; Timer dekrementieren
TIMER.I := TIMER.I - 1

; Ist unser Timer abgelaufen ?
IF TIMER.I <= 0 THEN

; Wir stoppen den Timer
Timer.Stop( 1 )

; Anzeige auf Null setzen
Label.Text( '00' )
  
```

```

; Timer für Blinken stoppen
; Timer.Stop( 2 )

; Blinkenden Stern löschen
; BLINK.I := 0

; Alarmton ausgeben
eI.P92_Pulse_Count      := 3
eI.P92_Time_ON         := 200
eI.P92_Time_OFF        := 100
InOut.DigitalOutputDriver( OP92, Output_Pulse )

; Nein, Timer läuft noch
ELSE

; Aktiver Timer in anderer Farbe darstellen
Label.Color( crimson )

; Wir löschen unseren Timer-Anzeige-String
TIMER.$ := ''

; Wir wollen eine führende Null bei Zeiten unter 10 Minuten
eI.FillChar := "0"

; Wir konvertieren den Timer-Stand in einen String
Str.Cvt_Integer( TIMER.$, TIMER.I, 2 )

; Wir setzen das Füllzeichen für Zahlenkonvertierungen wieder zurück
eI.FillChar := " "

; Wir zeichnen zuerst in den Hintergrund
Display.Prepare()

; Wir zeichnen den Timer-Stand
Label.Text( TIMER.$ )

; Wir kopieren das Gezeichnete in den Vordergrund
Display.ShowWindow()

ENDIF

ENDSUB

```

**Code-Erklärung:**

Der Alarmton wird durch Stromausgabe über den Output OP92 ausgelöst. OP92 ist mit einem Buzzer parallelgeschaltet. Im Beispiel sind 3 aufeinander folgende Töne von je 200 ms Länge mit einer Pause von 100 ms dazwischen. Das Programmieren von In- und Outputs mit eigerScript ist in den Application Notes eingehend erklärt.

Den Stopbefehl für den Timer 2 kommentieren wir vorerst noch aus, da er noch nicht installiert ist.

Ein Label kann nur String-Variablen schreiben. Deshalb mussten wir für das Label der Timer-Anzeige den Wert der Integer-Variablen Timer.I in den String Timer.\$ umwandeln. Diese String-Variable müssen wir noch bei den Variablen-Deklarationen definieren (Code 39). Gleichzeitig definieren wir auch noch die Integer-Variable `BLINK.I`, die wir für den blinkenden Stern brauchen werden.

**Code 39:** Deklaration der String-Variable für die Timer-Anzeige [mit max. 4 Zeichen] und einer Hilfsvariablen für einen blinkenden Stern.

```

STRING[ 4 ]   TIMER.$      ; der aktuelle Timer-Stand als String
INTEGER      BLINK.I = 0   ; Flag für den blinkenden Stern

```



Die Subroutine `SUB TIMER_1_EXPIRED` rufen wir auch aus den Down-Subroutinen der Vorwahlknöpfe auf (vgl. Code 18, S.11). So wird beim Drücken der Vorwahlknöpfe nicht nur der Timer gestartet, sondern es erscheint auch auf der Timer-Anzeige von Beginn an die gewählte Kochzeit. Dazu müssen wir noch bei allen 4 Down-Subroutinen (`SUB Select1_Down` etc.) die entsprechenden Subroutinen-Aufrufe auf „Uncomment“ setzen (vgl.

**Code 40:** Subroutinen-Aufrufe auf „Uncomment“ setzen. Betrifft die `SUB Select1_Down` bis `SUB Select4_Down`.

<code>; Vorwahl anzeigen</code> <code>; CallSubroutine( TIMER_1_EXPIRED )</code>	<code>; Vorwahl anzeigen</code> <code>CallSubroutine( TIMER_1_EXPIRED )</code>
---	---

## Step 16 Timer stoppen mit Stop-Button

Beim Stop-Button steht noch die Programmierung des Up-Events aus (Code 31).

Der Timer soll einerseits bei Ablauf der Kochzeit gestoppt werden (Code 38), andererseits aber auch mit dem Up\_Event des Stop-Buttons gestoppt werden können. Zudem hat der Up\_Event des Stop-Buttons auch eine allgemeine Reset-Funktion. Dadurch werden alle Vorwahlknöpfe wieder im Ruhezustand gezeichnet. All dies wird durch die Subroutine `SUB Stop_Up` geregelt (Code 41). Diese Subroutine enthält auch noch Code zum Stoppen des Blinker-Timers. Der ist jedoch noch nicht installiert. Deshalb kommentieren wir diese Zeilen vorerst noch aus.

**Code 41:** Subroutine für den Up-Event des Stop-Buttons. Der Code für den Blinker-Timer 2 wird vorerst noch auskommentiert.

```

; Wird einmal aufgerufen falls der Button 'STOP' losgelassen wird
SUB Stop_Up

; Timer für Kochzeit stoppen und Anzeige zurücksetzen
TIMER.I := 1
CallSubroutine( TIMER_1_EXPIRED )

; Timer für Blinken stoppen und Stern löschen

; Timer.Stop( 2 )
; BLINK.I := 0
; CallSubroutine( TIMER_2_EXPIRED )

; Alle Vorwahlen deaktivieren
CallSubroutine( Select1_Leave )
CallSubroutine( Select2_Leave )
CallSubroutine( Select3_Leave )
CallSubroutine( Select4_Leave )

CallSubroutine( Stop_Leave )

ENDSUB

```

Nun können wir in der HotSpot-Methode des Stop-Buttons für den Up-Event auf die Subroutine `SUB Stop_Up` verweisen (**Fehler! Verweisquelle konnte nicht gefunden werden.**).

**Code 42:** Der HotSpot des Stop-Buttons wird mit dem Aufruf der Subroutine `Stop_Up` ergänzt.

```
CallSubroutine( Stop_Leave )
HotSpot.Install( NIL, Stop_Leave, Stop_Down, Stop_Up )
```

## Step 17 Blinkender Stern

Im Vorlageprojekt blinkt ein Stern im Sekundentakt. Dieser Stern wird mit einem zweiten Timer gesteuert. Den Timer 2 programmieren wieder in den drei bekannten Schritten (vgl. Step 15.3, S.17)

### 17.1 Timer 2 installieren im Hauptprogramm

Den Timer 2 installieren wir so, dass der Stern je einmal pro Sekunde aufleuchtet und wieder verlöscht. Es muss also jede halbe Minute ein Wechsel stattfinden (Code 43).

**Code 43**

```
; Timer für Blinken installieren -----
Timer.InstallLocal( 2, TIMER_2_EXPIRED )
Timer.Load( 2, 500 ) ; 1000 msec = 1 Sekunde
```

### 17.2 Startbefehl für Timer 2

Der Timer 2 wird immer gleichzeitig mit dem Timer 1 gestartet, d.h. jedesmal beim Up-Event des Start-Buttons. Wir fügen den Startbefehl für den Timer 2 ebenfalls in die `SUB Start_Up` ein (Code 44).

**Code 44:** In der Subroutine für den Up-Event des Start-Buttons kommt der Startbefehl für den Timer 2 hinzu.

```
SUB Start_Up

; Timer für Kochzeit starten
Timer.StartContinuous( 1 )

; Timer für Blinken starten
Timer.StartContinuous( 2 )

; Wir zeichnen den Button wieder im inaktiven Zustand
CallSubroutine( Start_Leave )

ENDSUB
```

### 17.3 Anweisungen bei Ablauf von Timer 2

Alle halbe Sekunden läuft der Timer 2 ab, d.h. einmal pro Sekunde geht die Anweisung aus, das Stern-Symbol zu zeichnen, und das andere Mal soll der Stern wieder gelöscht werden.

Die Konstante für das Stern-Symbol ist `Symbol32A_Star8`. Die Symbol-Konstanten sind in der Header-Datei `DEF_eiger_Types_0$50.INC` aufgeführt. Das Symbol ist 32 x 32 Pixel gross und wird mit der Methode `Draw.SymbolNumber(Symbol32A_Star8)` gezeichnet. Dabei können wir auch die Farben des Symbols bestimmen.

Das alles formulieren wir in der `SUB TIMER_2_EXPIRED`, die vom Timer 2 jedesmal aufgerufen wird, wenn wieder ein Zeitzyklus abgelaufen ist (Code 45).

#### Code 45

```
; Blinkender Stern-Timer -----  
  
SUB TIMER_2_EXPIRED  
  
; Wir definieren das 'Fenster' in welchem der blinkende Stern angezeigt wird  
Load.Geometry_XYWH( 480, 125, 32, 32 )  
  
; Müssen wir den Stern löschen  
IF BLINK.I == 0 THEN  
  
; Wir kopieren das vorher definierte 'Fenster' aus dem Hintergrund (AVR)  
in den Vordergrund (RVR)  
Display.ShowWindow()  
  
; Nein, wir müssen ihn zeichnen  
ELSE  
  
; Wir wollen direkt in den Vordergrund-Bildspeicher (RVR) zeichnen  
Display.Direct()  
  
; Wir bestimmen die Farbe des Sterns  
eI.TextColor := yellow  
eI.FillColor := transparent  
eI.BackColor := transparent  
eI.LineColor := transparent  
  
; Wir zeichnen den Stern direkt in den Vordergrund-Bildspeicher  
Draw.SymbolNumber( Symbol32A_Star8 )  
  
ENDIF  
  
; Wir toggeln die Blink-Variable (aus 1 wird 0 bzw. aus 0 wird 1)  
BLINK.I := 1 - BLINK.I  
  
ENDSUB
```

### 17.4 Timer 2 stoppen

Das Blinken des Sterns soll einerseits bei Ablauf der Kochzeit aufhören, andererseits auch mit dem Stop-Button beendet werden können.



Dazu löschen wir die Semikolons bei den bisher auskommentierten Stopbefehlen in den Subroutinen `SUB Stop_Up` (vgl. Code 38,S.19) und `SUB TIMER_1_EXPIRED` (vgl. Code 41, S.21). → Code 46 und Code 47.

**Code 46:** Code-Zeilen für Timer 2 auf „Uncomment“ setzen. Betrifft die `SUB TIMER_1_EXPIRED`.

<pre>; Timer für Blinken stoppen ;   Timer.Stop( 2 )  ; Blinkenden Stern löschen ;   BLINK.I := 0</pre>	<pre>; Timer für Blinken stoppen       Timer.Stop( 2 )  ; Blinkenden Stern löschen       BLINK.I := 0</pre>
---	---

**Code 47:** Code-Zeilen für Timer 2 auf „Uncomment“ setzen. Betrifft die `SUB Stop_Up`.

<pre>; Timer für Blinken stoppen und Stern löschen ;   Timer.Stop( 2 ) ;   BLINK.I := 0 ;   CallSubroutine( TIMER_2_EXPIRED )</pre>	<pre>; Timer für Blinken stoppen und Stern löschen       Timer.Stop( 2 )       BLINK.I := 0       CallSubroutine( TIMER_2_EXPIRED )</pre>
---	---

## Step 18 Hintergrundbild laden

Am Schluss kommt noch die Krönung. Wir schmücken den Kitchen-Timer mit einem schönen Hintergrundbild. Das Bild „gemuese\_orig.jpg“ finden Sie im Verzeichnis „KTIM\PICT“. Wir formatieren das Bild mit Hilfe der `eigerGraphic Suite` um ins `EGI`-Format und speichern es im gleichen Verzeichnis als „gemuese.egi“.

Den Lade-Befehl für das Hintergrundbild positionieren wir im Hauptprogramm.

```
; Hintergrundbild laden

Load.Pos_X1Y1( 0, 0 )
File.Read_EGI( 'C:/KTIM/PICT/gemuese.egi' )
```

Nun ist das Projekt fertig. Wir kompilieren es und testen es auf dem `eigerPanel 57`.

**Herzliche Gratulation zum Gelingen Ihres ersten `eigerProjekts`!**

