12.2.9 Datenaustausch zwischen Anwendungsfenstern

Spätestens dann, wenn Sie aus einem (Haupt-)Fenster FMain heraus ein weiteres Fenster F1 starten, könnte die Frage zu beantworten sein, nach welchem Konzept der Daten-Transfer zwischen den zwei Anwendungsfenstern erfolgen soll. In das Konzept werden sicher die folgenden Überlegungen eingehen:

- Sind dem Fenster F1 über den Konstruktor Start-Werte mitzugeben?
- Erfolgt der Daten-Austausch zwischen FMain und F1 bidirektional oder unidirektional?
- Erfolgt der Daten-Austausch zwischen FMain und F1 einmalig oder mehrmalig, getaktet oder durch die Auswertung von Ereignissen?

In den folgenden Abschnitten werden Ihnen jeweils Kapitel genannt, in denen der Datentransfer für die o.a. Überlegungen schon beschrieben wurde.

12.2.9.1 Start-Werte F1

Start-Werte können Sie dem Fenster F1 mit einem entsprechend gestalteten Konstruktor über obligatorische und optionale Parameter in der _new-Methode mitgeben → Kapitel 12.2.7.1 Projekt 1. Eine andere Möglichkeit bieten selbst entwickelte Dialoge → Kapitel 24.3.4 Account.Manager.

12.2.9.2 Unidirektionaler oder bidirektionaler Daten-Austausch

- Selbst entwickelte Dialog-Fenster → Kapitel 12.2.4 Dialog-Fenster
- Persistente Fenster → Kapitel 12.2.6 Form – Persistenz (Projekt)

Sie können auch die Verschiebung von Daten aus Steuer-Elementen auf FMain und F1 über die Methoden Drag&Drop als Daten-Transfer auffassen.

12.2.9.3 Daten-Austausch – Projekt 1

In einem realen, erprobten Projekt wird die Temperatur über einen Temperatursensor auf einer Platine über eine RS232-Schnittstelle in einem Anwendungsfenster FSensor erfasst. Von einem Haupt-Fenster FMain aus wird auf den Temperatur-Messwert auf FSensor zugegriffen und die Temperatur in FMain angezeigt:

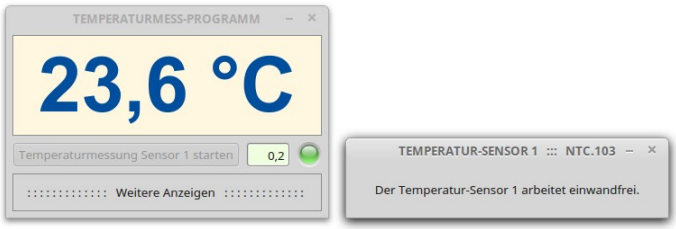

Abbildung 12.2.9.3.1: Temperaturanzeige auf FMain – daneben FSensor

Damit Sie das Projekt nachvollziehen können, werden die Temperatur-Werte über einen Zufallsgenerator in einem bestimmten Temperatur-Bereich erzeugt. Damit entfallen der Einsatz einer realen Platine und der Klasse SerialPort (gb.net).

Wenn Sie wechselseitig auf Eigenschaften und Methoden von Steuer-Elementen auf FMain und FSensor zugeifen wollen, dann müssen Sie neben der Deklaration projekt-weit geltender Variablen auch die Eigenschaft *Public* der betreffenden Steuer-Elemente bereits zur Entwicklungszeit auf True setzen. Zur Laufzeit ist das nicht möglich, weil die Eigenschaft *Public* virtuell ist. Wie bereits im → Kapitel 12.2.0 'Form – Klasse Window und Klasse Form' angemerkt gilt: *So lange es keinen zwingenden Grund gibt, die Public-Eigenschaft von False auf True zu ändern, sollten Sie der vollständigen Kapselung des Zustandes und des Verhaltens von Objekten den Vorzug geben.*

Andererseits gestaltet sich der bidirektionale Daten-Austausch zwischen FMain und FSensor über die öffentlichen Steuer-Elemente und Variablen sehr einfach. Der Quelltext für die Klassen FMain und FSensor in den Projekten 1 und 2 ist hinreichend kurz und wird deshalb vollständig angegeben.

FMain.class

```
' Gambas class file

Public Sub Form_Open()
  Application.MainWindow = FMain
  FMain.Center()
  FMain.Resizable = False
  piboxOn.Picture = Picture["LED/led_yellow.svg"]
  ValueBox1.Value = 0.2
  btnStartSensor1.Text = "Temperaturmessung Sensor 1 starten"
 End

Public Sub btnStartSensor1_Click()
  piboxOn.Picture = Picture["LED/led_green.svg"]
  FSensor.fTIBegin = 22.9
  FSensor.fTIEnd = 24
  FSensor.fTITrigger = ValueBox1.Value
  FSensor.Show()
  FSensor.X = FMain.X + FMain.W + 20
  FSensor.Y = FMain.Y + FMain.H - FSensor.H
  btnStartSensor1.Enabled = False
End

Public Sub ValueBox1_Change()
  If ValueBox1.Value >= 0 And If ValueBox1.Value <= 0.9 Then FSensor.fTITrigger = ValueBox1.Value
End

Public Sub Form_Close()
  FMain.Close()
End
```

FSensor.class

```
' Gambas class file

Public fCurTemperature As Float
Public fTIBegin As Float
Public fTIEnd As Float
Public fTITrigger As Float

Private fSavedTemperature As Float

Public Sub Form_Open()
  FSensor.Center()
  FSensor.Resizable = False
  TimerSimulation.Trigger()
End

Public Sub TimerSimulation_Timer()

  Dim bChanged As Boolean

  Randomize
  fCurTemperature = Rnd(fTIBegin, fTIEnd)
  bChanged = Abs(fCurTemperature - fSavedTemperature) > fTITrigger
  fSavedTemperature = fCurTemperature
  If bChanged Then
     FMain.lblTemperaturAnzeige.Text = Format(FSensor.fCurTemperature, "##.0 °C")
  Endif
End

Public Sub Form_Close()
  FMain.piboxOn.Picture = Picture["LED/led_yellow.svg"] ' piboxOn.Public = True
  FMain.btnStartSensor1.Enabled = True                  ' btnStartSensor1.Public = True
  FMain.lblTemperaturAnzeige.Text = "--- °C"           ' lblTemperaturAnzeige.Public = True
  FMain.ValueBox1.Value = 0.2                           ' ValueBox1.Public = True
End
```

Kommentar:

- Die Anzeige der Temperatur ändert sich nur dann, wenn die Temperatur-Differenz größer ist als der Wert der Variablen fTITrigger.
- Die Klasse FSensor enthält wegen der engen Bindung an die Klasse FMain keinen wieder-verwendbaren Quelltext, wie es besonders die letzten Zeilen in FSensor.class deutlich zeigen; der Quelltext wird schwer lesbar.
- Wenn sich zum Beispiel der Name der Klasse *FMain* ändert, dann sind diverse Änderungen in *beiden* Klassen notwendig.

12.2.9.4 Daten-Austausch – Projekt 2

Im zweiten Projekt erfolgt der *Daten-Austausch zwischen den Anwendungsfenstern* FMain und FSensor nur über *öffentliche* Schnittstellen, deren Signaturen bekannt sind. Jede Klasse kapselt somit ihre Daten vollständig gegenüber der anderen Klasse. Das wird auch im Quelltext für die beiden Klassen FMain und FSensor deutlich: Es gibt keine öffentlichen Steuer-Elemente und auch keine öffentlichen Variablen!

FMain.class

```
' Gambas class file

Private fTITrigger As Float = 0.2
Public hSensor As FSensor

Public Sub Form_Open()
  Application.MainWindow = FMain
  FMain.Center()
  FMain.Resizable = False
  piboxOn.Picture = Picture["LED/led_yellow.svg"]
  ValueBox1.Value = fTITrigger
 End

 Public Sub btnStartSensor1_Click()
  CreateSensor1()
  piboxOn.Picture = Picture["LED/led_green.svg"]
  btnStartSensor1.Enabled = False
End

Public Sub Sensor1_Close()
  piboxOn.Picture = Picture["LED/led_yellow.svg"]
  btnStartSensor1.Enabled = True
  lblTemperature1.Text = "--- °C"
  ValueBox1.Value = fTITrigger
End

Public Sub Sensor1_Change()
  lblTemperature1.Text = Format(hSensor.Temperature, "##.0 °C")
End

Public Sub ValueBox1_Change()
  If hSensor Then
     If ValueBox1.Value >= 0 And If ValueBox1.Value <= 0.9 Then
        hSensor.TriggerValue = ValueBox1.Value
     Endif
  Endif
End

Public Sub CreateSensor1()

  Dim hObserver As Observer
  Dim aRanges As Float[] = [19.9, 21]

  hSensor = New FSensor(aRanges[0], aRanges[1])
  hSensor.Caption = "TEMPERATUR-SENSOR 1"
  hSensor.TriggerValue = fTITrigger
  hSensor.Show()
  hSensor.X = FMain.X + FMain.W + 20
  hSensor.Y = FMain.Y + FMain.H - hSensor.H
  hObserver = New Observer(hSensor) As "Sensor1"
End

Public Sub Form_Close()
  FMain.Close()
End
```

FSensor.class

```
' Gambas class file

Private fSavedTemperature As Float = 0.0
Private $fIBegin As Float
Private $fIEnd As Float
'-------------------------------------------------------------------
'' Eigenschaft: Temperature, Zugriffsrecht: public, Modus: read-only
Property Read Temperature As Float
'' Eigenschaft: TriggerValue, Zugriffsrecht: public
Property TriggerValue As Float
'-------------------------------------------------------------------
Private $fTemperature As Float     ' Lokale Variable ↘ Schnitt-Stelle
```

```
Private $fTriggerValue As Float    ' Lokale Variable ↘ Schnitt-Stelle
'-------------------------------------------------------------------
'' Ereignis: Change
Event Change
'-------------------------------------------------------------------

Public Sub _new(IBegin As Float, IEnd As Float)
  If Not TypeOf(IBegin) = gb.Float Then Error.Raise("Typ-Fehler: Startwert von Temperatur-Intervall.")
  If Not TypeOf(IEnd) = gb.Float Then Error.Raise("Typ-Fehler: Endwert von Temperatur-Intervall.")
  If IBegin >= IEnd Then Error.Raise("Der Startwert ist größer oder gleich dem Endwert vom Intervall.")
  $fIBegin = IBegin
  $fIEnd = IEnd
End

Public Sub Form_Open()
  Me.Center()
  Me.Resizable = False
  TimerSimulation.Delay = 1000
  TimerSimulation.Start()
  TimerSimulation.Trigger()
End ' Form_Open()

Public Sub TimerSimulation_Timer()

  Dim bChanged As Boolean

  Randomize
  $fTemperature = Rnd($fIBegin, $fIEnd)
  bChanged = Abs($fTemperature - fSavedTemperature) > $fTriggerValue
  fSavedTemperature = $fTemperature

  If bChanged Then Raise Change

End

' Zusätzlicher Dienst oder Schnittstelle (öffentliche Methode): Temperatur_lesen
Private Function Temperature_Read() As Float
  Return $fTemperature
End

' Zusätzlicher Dienst oder Schnittstelle (öffentliche Methode): Trigger_lesen
Private Function TriggerValue_Read() As Float
  Return $fTriggerValue
End

' Zusätzlicher Dienst oder Schnittstelle (öffentliche Methode): Trigger_schreiben
Private Sub TriggerValue_Write(Value As Float)
  If Value < 0 Or Value > 1 Then
     Error.Raise("Der Trigger-Wert liegt nicht im Intervall [0|1]")
  Endif

  $fTriggerValue = Value

End
```

Kommentar:

- Wenn Sie die Interaktion von FMain und FSensor betrachten können Sie feststellen, dass FSensor Daten produziert und durch das Ereignis 'Change' FMain nur benachrichtigt, wenn neue Daten verfügbar sind. FMain liest über eine öffentliche Schnittstelle mit bekannter Signatur den Temperaturwert als ReadOnly-Eigenschaft von FSensor aus und bereitet ihn auf, um ihn im Steuer-Element *lblTemperature1* anzuzeigen. Das ist vorbildliche Daten-Kapselung: Kein Objekt weiß vom anderen mehr, als es unbedingt sein muss und jedes kümmert sich um seine eigenen Aufgaben!

```
Public Sub TimerSimulation_Timer()
  Dim bChanged As Boolean

  Randomize
  $fTemperature = Rnd($fIBegin, $fIEnd)
  bChanged = Abs($fTemperature - fSavedTemperature) > $fTriggerValue
  fSavedTemperature = $fTemperature
  If bChanged Then Raise Change

End
```

```
Public Sub Sensor1_Change()
  lblTemperature1.Text = Format(hSensor.Temperature, "##.0 °C")
End
```

- In gleicher Weise wird FMain über das Ereignis 'Sensor1_Close()' informiert, dass das Anwendungsfenster FSensor geschlossen wurde. Wie die Klasse FMain auf dieses Ereignis reagiert wird allein im Quell-Text der Klasse FMain festgelegt!

```
Public Sub Sensor1_Close()
  piboxOn.Picture = Picture["LED/led_yellow.svg"]
  btnStartSensor1.Enabled = True
  lblTemperature1.Text = "--- °C"
  ValueBox1.Value = fTITrigger
End
```

- Die Eigenschaft *FSensor.TriggerValue* kann nur über eine öffentliche Schnittstelle gelesen und geschrieben werden.
- Die Klasse FSensor verfügt über einen Konstruktor mit zwei Parametern, die im vorliegenden Projekt nur den Temperatur-Bereich festlegen, in dem die Temperatur-Werte zufällig erzeugt werden:

```
FSensor.class:
Public Sub _new(IBegin As Float, IEnd As Float)
  If Not TypeOf(IBegin) = gb.Float Then Error.Raise("Typ-Fehler: Startwert.")
  If Not TypeOf(IEnd) = gb.Float Then Error.Raise("Typ-Fehler: Endwert von Temperatur-Intervall.")
  If IBegin >= IEnd Then Error.Raise("Der Startwert ist größer oder gleich dem Endwert).")
  $fIBegin = IBegin
  $fIEnd = IEnd
End

FMain.class:
Public Sub CreateSensor1()
  Dim hObserver As Observer
  Dim aRanges As Float[] = [19.9, 21]

  hSensor = New FSensor(aRanges[0], aRanges[1])
  hSensor.Caption = "TEMPERATUR-SENSOR 1"
  hSensor.TriggerValue = fTITrigger
  hSensor.Show()
  hSensor.X = FMain.X + FMain.W + 20
  hSensor.Y = FMain.Y + FMain.H - hSensor.H
  hObserver = New Observer(hSensor) As "Sensor1"
End
```

  Der Observer muss erzeugt werden, da die Klasse FSensor zwar von der Klasse Form erbt, aber nicht die Erweiterungen (2 Eigenschaften und ein Ereignis) der Klasse 'im Blick' hat.

Das Projekt bietet m.E. mehrere Vorteile gegenüber Projekt 1:

- Sie können die Steuerelemente auf FMain problemlos umbenennen oder allgemeiner gesagt die Implementierung der Temperaturdarstellung vollständig ändern, ohne im Hinterkopf haben zu müssen, welche Klassen sich auf die aktuellen Implementierungsdetails verlassen und diese dann entsprechend zu ändern.
- Der interne Code der Klasse FSensor ist leichter verständlich, weil er in sich geschlossen ist.
- Der Zustand eines Objekts der Klasse ist konsistenter, wenn der Zugriff von außen nur durch die öffentlichen Schnittstellen geregelt ist. Anders gesagt: Wenn weniger nach außen sichtbar ist, können auch die Schnittstellen weniger falsch benutzt werden.

Die Vorteile klingen überzeugend. Nur werden Sie recht bald feststellen, dass diese Vorteile bei größeren Projekten schnell dahin schmelzen, denn eine Klasse konsequent und ordentlich zu kapseln kann im Design sehr schwierig und langwierig sein. Gelegentlich wird der Code sogar zu umständlich. Nur das sind die Fälle für die sorgfältig abgewogene Verwendung öffentlicher Steuerelemente! So ein Fall liegt für das Projekt 1 nicht vor, so dass dem Projekt 2 der Vorzug gegeben werden sollte.

12.2.9.5 Daten-Austausch – Projekt 3

Das 3. Projekt arbeitet mit 3 Sensor-Anwendungsfenstern. Alle drei Anwendungsfenster sind Instanzen der Klasse FSensor, die aus dem Projekt 2 übernommen wird. Es wird deshalb nur der Quelltext der Klasse FMain vorgestellt und kommentiert:

```
[1]  ' Gambas class file
[2]
```

```
[3]  Private $iSensors As Integer = 1
[4]  Private aSensors As New FSensor[]
[5]  Private aTrigger As Float[] = [0.1, 0.3, 0.5]
[6]
[7]  Public Sub Form_Open()
[8]    Application.MainWindow = FMain
[9]    FMain.Center
[10]   FMain.Resizable = False
[11]   piboxOn.Picture = Picture["LED/led_green.svg"]
[12]   CreateSensors()
[13]   ValueBox1.Value = aTrigger[0]
[14]   ValueBox2.Value = aTrigger[1]
[15]   ValueBox3.Value = aTrigger[2]
[16] End
[17]
[18] Private Sub CreateSensors()
[19]   Dim hSensor As FSensor, hObserver As Observer
[20]   Dim i As Integer
[21]   Dim aRanges As Float[][] = [[20, 21], [40.0, 42], [33, 35.8]]
[22]
[23]   For i = 1 To aRanges.Count
[24]     hSensor = New FSensor(aRanges[i - 1][0], aRanges[i - 1][1])
[25]     hSensor.Caption = "TEMPERATUR-SENSOR " & Str($iSensors)
[26]     hSensor.Tag = $iSensors ' Die Tag-Eigenschaft speichert die Nummer des Sensors
[27]     hSensor.TriggerValue = aTrigger[i - 1]
[28]     hSensor.Show()
[29]     aSensors.Add(hSensor)
[30]     hObserver = New Observer(aSensors[i - 1]) As "Sensor"
[31]     hObserver.Tag = $iSensors ' Die Tag-Eigenschaft speichert die Nummer des Observers
[32]     Inc $iSensors
[33]   Next
[34] End
[35]
[36] Public Sub Sensor_Change()
[37]   Select Case Last.Tag
[38]     Case 1
[39]       lblTemperature1.Text = Format(Last.Temperature, "##.0 °C")
[40]     Case 2
[41]       lblTemperature2.Text = Format(Last.Temperature, "##.0 °C")
[42]     Case 3
[43]       lblTemperature3.Text = Format(Last.Temperature, "##.0 °C")
[44]   End Select
[45] End
[46]
[47] Public Sub Sensor_Close()
[48]   Select Case Last.Tag
[49]     Case 1
[50]       lblTemperature1.Text = "--- °C"
[51]     Case 2
[52]       lblTemperature2.Text = "--- °C"
[53]     Case 3
[54]       lblTemperature3.Text = "--- °C"
[55]   End Select
[56] End
[57]
[58] Public Sub ValueBox1_Change()
[59]   aSensors[0].TriggerValue = ValueBox1.Value
[60] End
[61]
[62] Public Sub ValueBox2_Change()
[63]   aSensors[1].TriggerValue = ValueBox2.Value
[64] End
[65]
[66] Public Sub ValueBox3_Change()
[67]   aSensors[2].TriggerValue = ValueBox3.Value
[68] End
```

Kommentar:

- Die Zeile 8 bietet den Vorteil, dass beim Schließen von FMain alle von FMain geöffneten Fenster ebenfalls geschlossen werden.
- Die Anzahl der Temperaturbereiche in der Zeile 21 bestimmt die Anzahl der Instanzen der Klasse FSensor.
- In der Zeile 24 wird eine neue Instanz der Klasse FSensor erzeugt, der 2 Parameter übergeben werden.
- Beachten Sie, dass alle Observer den gleichen Event-Namen 'Sensor“ besitzen, unter dem Ereignisse ausgelöst werden!
- Das Speichern der Nummer des Sensors und des Observers in der Tag-Eigenschaft (Zeile 26 und Zeile 31) bietet den Vorteil, auf die beiden Ereignisse *Sensor.Change()* und *Sensor.Close()*

in den Zeilen 36 bis 45 und in den Zeilen 47 bis 56 kompakt zu reagieren. Dazu müssen Sie nur die Referenz auf das Objekt ermitteln, das das letzte Ereignis ausgelöst hat. Das gelingt mit der Kombination aus *Last.Tag* und *Last.Temperature*, denn Last gibt eine Referenz auf das Objekt zurück, das das letzte Ereignis ausgelöst hat.

Hinweis: Eine Alternative bietet sich, wenn Sie die Zeile 30 durch diesen Quelltext ersetzen:

```
hObserver = New Observer(aSensors[i - 1]) As "Sensor" & Str($iSensors)
```

Nun erhalten Sie drei Observer mit jeweils eigenem Event-Namen und können aus diesem Grund das Change()-Ereignis und das Close()-Ereignis separat auswerten, wenn Sie die Zeilen 36 bis 56 durch die folgenden beiden Quelltext-Abschnitte ersetzen:

```
Public Sub Sensor1_Change()
  lblTemperature1.Text = Format(aSensors[0].Temperature, "##.0 °C")
End

Public Sub Sensor2_Change()
  lblTemperature2.Text = Format(aSensors[1].Temperature, "##.0 °C")
End

Public Sub Sensor3_Change()
  lblTemperature3.Text = Format(aSensors[2].Temperature, "##.0 °C")
End
```

```
Public Sub Sensor1_Close()
  lblTemperature1.Text = "--- °C"
  ValueBox1.Enabled = False
End

Public Sub Sensor2_Close()
  lblTemperature2.Text = "--- °C"
  ValueBox2.Enabled = False
End

Public Sub Sensor3_Close()
  lblTemperature3.Text = "--- °C"
  ValueBox3.Enabled = False
End
```