16.6.2 Valide Daten

In allen Computerprogrammen werden Daten verarbeitet. Daten sind dabei als an Zeichen gebundene Informationen zu sehen. Diese Aussagen stellen noch einmal klar, dass wir es gegenwärtig immer noch mit elektronischer Datenverarbeitung (EDV) zu tun haben! Heute geschieht das auf einem hohen Niveau in Bezug auf die verwendeten *Algorithmen*, *Programmiersprachen* und die eingesetzten *Computer*. Kein Vergleich zu den Anfängen in der EDV, die der Autor im Rechenzentrum an der TH Magdeburg und in den EDV-Vorlesungen von Dr. Stuchlik von 1967-1971 erlebte.

Die zu verarbeitenden Daten

- werden entweder über (externe) Datenspeicher eingelesen oder
- in Echtzeit von Sensoren über geeignete Interfaces bereitgestellt oder
- entstehen als (temporäre) Daten - zum Beispiel als Ergebnis von Berechnungen - zur Laufzeit des Programms oder
- sie werden in Texteingabe-Komponenten wie zum Beispiel TextBox oder ValueBox über eine (reale oder virtuelle) Tastatur als Zeichenkette ausgelesen, konvertiert und geprüft.

In diesem Kapitel steht die Komponente 'Textbox' im Fokus. Bedenken Sie, dass eine Textbox in der Eigenschaft *TextBox.Text* eine Zeichenkette (Datentyp String) zurück gibt, wenn Sie Eingaben aus einer TextBox auslesen. Programmspezifisch ist zu entscheiden, ob Sie den Text aus einer Textbox

- als normalen Text (Name, Ort, Beschreibung, ...)
- als formatierten Text (Geld-Wert, Datum, MAC-Adresse, Bestellnummer, Farbwerte, ...) oder
- als Zahl zum Zählen (Anzahl mit $z \in \mathbb{N}$; $z \geq 0$ oder $z \geq 1$) oder
- als Zahl zum Rechnen (Messwert, Parameter, Funktionswert, ...)

weiterverwenden wollen.

Unabhängig von Ihrer Entscheidung für den konkreten Einsatzfall sollten Sie als Programm-Entwickler dafür Sorge tragen, dass nur zulässige Zeichenketten als (valide) Daten aus einer TextBox an das Programm weitergereicht werden, um Laufzeitfehler zu verhindern! Ein Laufzeitfehler im Programmablauf entsteht zum Beispiel dann, wenn in eine TextBox eine Zeichenkette eingegeben wird, die sich nicht in eine natürliche Zahl konvertieren lässt, die dort erwartet wird oder in einer Prozedur auf eine Datei zugegriffen wird, die im angegebenen Verzeichnis nicht existiert. Eine Vielzahl von solchen Fehlern kann man durch eine defensive Programmierung abfangen, anzeigen oder in geeigneten Fehlerbehandlungsroutinen bearbeiten. In allen Fällen gilt: Der Schutz des Programms hat Vorrang!

Bevor Sie sich auf den durchaus beschwerlichen Weg machen, um Eingaben aus einer Textbox zu validieren, sollten Sie prüfen, ob es Alternativen zur TextBox und deren Spezialisierungen gibt, um *sichere Eingaben* vornehmen zu können!

Zu diesen Alternativen - im Kontext mit den zu lösenden Aufgaben - gehören die folgenden Komponenten:

- ComboBox, wenn deren Eigenschaft ReadOnly auf True gesetzt wurde
- Listbox
- SpinBox
- Slider
- Dial
- ColorButton
- Alle Chooser (Farbe, Datum, Verzeichnis, Dateien, Font)

Die folgenden Komponenten gehören <u>nicht</u> zu den Alternativen, weil Sie die Eingaben *in jedem Fall* prüfen müssen, um valide Daten weiter verarbeiten zu können:

- InputBox
- MaskBox
- ValueBox
- ButtonBox
- DateBox

Kommt keine der oben genannten alternativen Komponenten für Ihr Gambas-Projekt in Betracht, sind für die Validierung von Eingaben aus einer TextBox (InputBox, MaskBox, ValueBox, HistoryBox und ButtonBox) eigene Konzepte zu entwerfen, umzusetzen und zu testen. Ein erprobter Ansatz besteht darin

1. unmittelbar nach der Eingabe jedes einzelnen Zeichens zu testen, ob das eingegebene Zeichen in der Menge der zulässigen Zeichen – dem Eingabe-Alphabet – enthalten ist oder nicht und
2. nach dem Abschluss der Eingabe zu prüfen, ob die eingegebene Zeichenkette ein Wort aus der (formalen) Sprache der zulässigen Eingaben ist.

Oft wird auf den Zeichen-Scanner als Vor-Prüfung verzichtet und nur der Rückgabewert aus der Textbox oder ihren Spezialisierungen untersucht. Es ist zu beachten, dass der Rückgabewert aus einer TextBox, MaskBox oder HistoryBox vom Typ String und vom Typ Date aus einer DateBox ist.

16.6.2.1 Aufgaben

Die folgenden Aufgaben sollen bearbeitet werden, um Eingaben aus einer TextBox oder deren Spezialisierungen zu prüfen, um valide Daten bereitzustellen:

(A1) Sichern Sie, dass in einem Programm nur valide Datumswerte im (deutschen) Format tt.mm.jjjj weiter verarbeitet werden!
(A2) Entwickeln Sie geeignete Prozeduren, um Eingaben aus den folgenden Komponenten zu prüfen:

- Datum in einem speziellen Format → TextBox. Es ist zu prüfen, ob in einer TextBox ein Datum im speziellen (deutschen) Format: tt.<leerzeichen>Monatsname<leerzeichen>jjjj steht?
- MAC-Adresse → MaskBox
- Datum → DateBox
- IP-Adresse → ValueBox (Typ IPAddress)

Lösung Aufgabe A1:

Für die Lösung dieser Aufgabe gibt es nur die Komponente *DateChooser* im Modus 0 (*DateOnly)* als Alternative zur TextBox und deren Spezialisierungen, weil nur Daten ausgewählt werden können, die ein <u>valides</u> Datum (→ Abbildung 16.6.1.1.1; rechts) repräsentieren:

```
Public dDatum As Date
...
Public Sub DateChooser1_Activate()
  dDatum = DateChooser1.Value
  DateBox1.Value = dDatum
End ' DateChooser1_Activate()
```

Lösungsansätze Aufgabe A2:

Im Falle des Einsatzes einer *MaskBox* oder *DateBox* oder *ValueBox* (Typ Datum) müssen Sie selbst die Eingaben prüfen, bevor Sie ein eingegebenes Datum in der Variable *dDatum* speichern und weiter verarbeiten können oder eine Fehlermeldung anzeigen lassen:

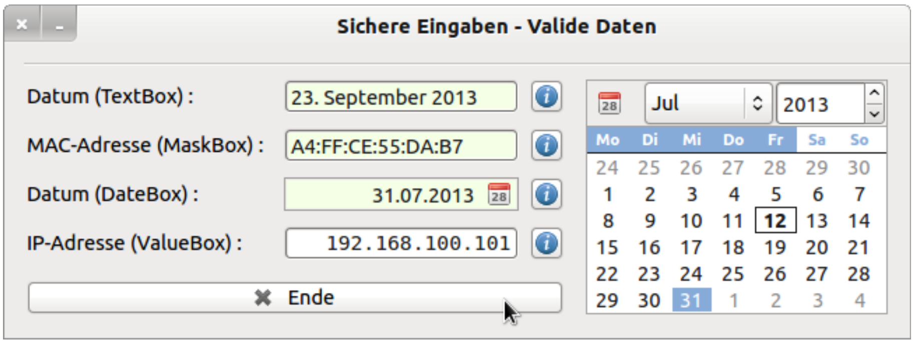

Abbildung 16.6.2.1.1: Sichere Eingaben aus einer TextBox oder deren Spezialisierungen

Lösung Teilaufgabe A.2.1:

In der Datum-TextBox wird die Funktion *IsDate()* für die Prüfung des Datums im deutschen Format *tt.mm.jjjj* verwendet. Diese Funktion bezieht jedoch die aktuelle System-Sprache in ihren Test ein. Daher kann ein Datum im deutschen Format nur als solches erkannt werden, wenn die System-Sprache auf *deutsch* eingestellt ist, sonst liefert das Programm trotz formal richtiger Datumseingabe immer einen Eingabefehler.

Die verwendete Lösungsidee besteht darin

- zuerst die Zeichenkette aus der TextBox in der lokalen Variablen *sEingabe* (Zeile 24) speichern,
- dann alle Leerzeichen am Anfang und am Ende zu entfernen (Zeile 25) sowie alle doppelten Leerzeichen in der Zeichenkette auf ein Leerzeichen zu reduzieren, falls diese mehrfachen Leerzeichen existieren (Zeilen 27 bis 29) und
- dann die Zeichenkette mit der Split-Funktion in 3 Teile zu zerlegen (Zeile 32), wobei das Leerzeichen als Trennzeichen verwendet wird. Alle 3 Teile werden als Element in einem String-Array aMatrix gespeichert.

Diese Idee lässt sich für viele Aufgabenstellungen mit anderen Trennzeichen adaptieren. Der Quelltext für alle vier *Teillösungen* wird vollständig angegeben und kommentiert:

```
[1] Public dDatum As Date
[2] Public sMAC As String
[3] Public sMessage As String
[4] Public sIPAdresse As String
[5]
[6] Public Sub Form_Open()
[7]   FMain.Center()
[8]   FMain.Resizable = False
[9] ' Mode-Konstanten stehen in der DateChooser-Klasse und nicht in DateBox-Klasse!
[10]  DateBox1.Mode = DateChooser.DateOnly
[11]' DateBox1.Value = Now ' Dann wäre keine Datenprüfung notwendig!
[12]  DateBox1.ReadOnly = True
[13]  ValueBoxIP.Type = ValueBoxIP.IPAddress
[14]  MaskBoxMAC.Mask = "[a-fA-F0-9][a-fA-F0-9]:[a-fA-F0-9][a-fA-F0-9]:[a-fA-F0-9] \
             [a-fA-F0-9]:[a-fA-F0-9][a-fA-F0-9]:[a-fA-F0-9][a-fA-F0-9]:[a-fA-F0-9][a-fA-F0-9]"
[15]End ' Form_Open()
[16]
[17]Public Sub txtDate_Activate()
[18]  Dim aMatrix As String[]
[19]  Dim aMonate As String[]
[20]  Dim sMonat, sEingabe, sDatum As String
[21]
[22]  aMonate = ["Januar", "Februar", "März", "April", "Mai", "Juni", "Juli", "August", \
                 "September", "Oktober", "November", "Dezember"]
[23]
[24]  sEingabe = txtDate.Text
[25]  sEingabe = Trim(sEingabe)
[26]
[27]  While InStr(sEingabe, "  ")
[28]    sEingabe = Replace$(sEingabe, "  ", " ")
[29]  Wend
[30]  txtDate.Text = sEingabe
[31]
[32]  aMatrix = Split(sEingabe, " ")
[33]
[34]  If aMatrix.Count <> 3 Then
[35]     Message.Error("EINGABEFEHLER")
[36]     Return
[37]  Endif ' aMatrix.Count <> 3 ?
[38]
[39]  If aMonate.Exist(aMatrix[1]) Then
[40]     sMonat = Str(aMonate.Find(aMatrix[1]) + 1)
[41]  Else
[42]     Message.Error("EINGABEFEHLER")
[43]     Return
[44]  Endif ' aMonate.Exist(aMatrix[1]) ?
[45]
[46]  sDatum = aMatrix[0] & sMonat & "." & aMatrix[2]
```

```
[47]
[48]  If IsDate(sDatum) Then
[49]     dDatum = Val(sDatum)
[50]     DateBox1.Value = dDatum ' Zur Kontrolle
[51]  Else
[52]     Message.Error("EINGABEFEHLER")
[53]     Return
[54]  Endif ' IsDate(sDatum)
[55]End ' txtDate_Activate()
```

Kommentar:

- In den Zeilen 34-37 wird geprüft, ob genau 3 Elemente im Array *aMatrix* gespeichert wurden. Im Fehlerfall wird eine Fehler-Meldung ausgegeben.
- Die Prüfung, ob der Monatsname korrekt geschrieben wurde erfolgt in den Zeilen 39-44. Tritt ein Fehler auf, wird eine entsprechende Meldung ausgegeben. Bei positivem Test wird die Variable *sMonat* in der Zeile 46 genutzt, um einen Datum-String im Format tt.mm.jjjj zusammenzufügen.
- Abschließend erfolgt die Prüfung mit der Funktion IsDate(...) , ob die Zeichenkette sDatum als valides deutsches Datum interpretiert werden kann – sonst gibt es eine passende Meldung.

Lösung Teilaufgabe A.2.2

"*Die MAC-Adresse (Media-Access-Control-Adresse) ist die Hardware-Adresse jedes einzelnen Netzwerkadapters, die als eindeutiger Identifikator des Geräts in einem Rechnernetz dient.*" findet man unter dem Link: *http://de.wikipedia.org/wiki/MAC-Adresse*. Eine MAC-Adresse ist ein 48-Bit-Datenwort oder ein 6-Byte-Datenwort. Die 6 Byte werden hexadezimal angegeben und werden durch typisch 5 Doppelpunkte voneinander getrennt. Es wird eine angepasste Lösungsidee wie in der Teilaufgabe A.2.1 umgesetzt. Es ist durch die Verwendung der gesetzten MAC-Maske garantiert, dass nur zulässige Zeichen [0-9a-fA-F] von der MaskBox akzeptiert werden. Alle anderen Zeichen werden ignoriert. Daher könnten die Zeilen 10 bis 13 und ein Zeichen-Scanner in den Zeilen 15 bis 20 hier auch entfallen. Die Maske mit den Leerstellen und den Doppelpunkten wird erst angezeigt, wenn Sie die Box mit einem Einfach-Klick aktivieren. Der Quelltext in der Zeile 22 sorgt nur dafür, dass die eingegebenen Buchstaben in der MAC-Adresse als Großbuchstaben angezeigt werden – reine Anzeigekosmetik.

```
[1] Public Sub MaskBoxMAC_Activate()
[2] ' Datentyp von MaskBoxMAC.Text ist String
[3]   Dim aMatrix As String[]
[4]   Dim sEingabe, sElement As String
[5]
[6]   sEingabe = MaskBoxMAC.Text
[7]   sEingabe = Trim(sEingabe)
[8]   aMatrix = Split(sEingabe, ":")
[9]
[10]  If aMatrix.Count <> 6 Then
[11]     Message.Error("EINGABEFEHLER_1")
[12]     Return
[13]  Endif ' aMatrix.Count <> 6 ? ' Kann hier entfallen, da eine Maske eingesetzt wird
[14]
[15]  For Each sElement In aMatrix
[16]      If Left(sElement) Not Like "[abcdefABCDEF0123456789]" Or   \
             Right(sElement) Not Like "[abcdefABCDEF0123456789]" Then
[17] 'If Left(sElement) Not Like "[a-fA-F0-9]" Or Right(sElement) Not Like "[a-fA-F0-9]" Then
[18]         Message.Error("EINGABEFEHLER_2")
[19]         Return
[20]      Endif ' Zeichen nicht im Eingabe-Alphabet ?
[21]  Next ' sElement
[22]  sMAC = Upper(sEingabe)
[23]  MaskBoxMAC.Text = sMAC
[24]End ' MaskBoxDate_Activate()
```

Lösung Teilaufgabe A.2.3

Eine DateBox ist eine spezialisierte MaskBox. Durch die Angabe der Eigenschaft *DateBox.Mode* können Sie entweder ein Datum (Mode = 0) auswählen oder eine Zeit (Mode = 2) oder einen Zeitstempel mit Datum und Zeit (Mode = 1). Die Mode-Konstanten stehen gegenwärtig nur in der DateChooser-Klasse und <u>nicht</u> in der DateBox-Klasse zur Verfügung – was zu erwarten wäre. Sie müssen den Modus so festlegen:

```
DateBox1.Mode = 0
DateBox1.Mode = DateChooser.DateOnly ' Alternative
```

Mit einem Klick auf den kleinen Button rechts öffnet sich ein Datum-Auswahl-Dialog, aus dem Sie ein valides Datum auswählen können, dass dann mit der entsprechenden Datumsmaske angezeigt wird. Setzen Sie die Eigenschaft *DateBox.Readonly* auf True, dann können Sie dieses angezeigte Datum nicht mehr verändern.

Initialisieren Sie die DateBox nicht mit einem (validen) Datum, dann müssen Sie die Eingaben in die DateBox so prüfen:

```
Public Sub DateBox1_Activate()
' TypeOf(DateBox1.Value) = 8 => Datentyp: Date

  If Not IsDate(Str(DateBox1.Value)) Then
     sMessage = "Die Eingabe ist kein korrektes Datum!" ' DateBox1.Value ist dann NULL!
     Message.Error(sMessage)
     Return
  Else
     dDatum = DateBox1.Value
  Endif ' Not IsDate(Str(DateBox1.Value)) ?

End ' DateBox1_Activate()

Public Sub DateBox1_DblClick()
  DateBox1_Activate()
End ' DateBox1_DblClick()
```

Mit den folgenden Zeilen in der Prozedur Form_Open() entfällt eine Prüfung. Sie können dann aber kein Datum mehr selbst eintragen und müssen den Datum-Auswahl-Dialog nutzen – erhalten dann aber valide Datum-Daten:

```
DateBox1.Mode = DateChooser.DateOnly
DateBox1.Value = Now
DateBox1.ReadOnly = True
```

Lösung Teilaufgabe A.2.4

Die Eingabe von IP-Adressen in eine Value-Box als spezialisierter MaskBox gelingt problemlos mit der Vorgabe des Typs in der Prozedur Form_Open() mit:

```
ValueBoxIP.Type = ValueBoxIP.IPAddress
```

Sie müssen die eingegebene IP-Adresse aber prüfen, weil die Vorgabe der Maske zwar nur Ziffern an den zulässigen Stellen in den 4 Ziffernblöcken als Eingabezeichen zulässt – aber diese an jeder Position im Bereich von 0 bis 9. Damit werden sowohl 192.189.100.1 als auch 192.189.245.202 als Eingaben akzeptiert! Die Prüfung der Eingaben auf valide Daten greift auf erprobte Ansätze zurück:

```
Public Sub ValueBoxIP_DblClick()
' Datentyp von ValueBoxIP.Value ist String
  Dim aMatrix As String[]
  Dim iCount As Integer

  aMatrix = Split(ValueBoxIP.Value, ".")
  For iCount = 0 To aMatrix.Max
    If Val(aMatrix[iCount]) < 0 Or Val(aMatrix[iCount]) > 255 Then
       sMessage = "Die IP-Adresse ist im " & (icount + 1) & ". Block fehlerhaft (" & \
                   aMatrix[iCount] & ") !"
       Message.Error(sMessage)
       Return
    Endif ' Fehler ?
  Next ' iCount
  sIPAdresse = ValueBoxIP.Value
End ' ValueBoxIP_DblClick()
```

Nach der Eingabe einer IP-Adresse erfolgt die Prüfung mit einem *Doppelklick* auf die ValueBox. Beachten Sie, dass Sie die Position des Cursors in der ValueBox mit den Pfeil-Tasten < oder > ändern! Hinweis:

Bitte beachten Sie unbedingt die Ausführungen im Kapitel → 19.6.5 Prüfung der Syntax von Zeichenketten, weil mit dem neuen Operator MATCH (ab Gambas 3.4.2) in der Komponente gb.pcre (Perl Compatible Regular Expression) die Prüfung von Zeichenketten wesentlich vereinfacht wird.