19.6.5 Prüfung der Syntax von Zeichenketten

Vor allem in Formularen müssen die als Zeichenketten eingegebenen Daten intensiv geprüft werden, damit nur valide Daten weiterverarbeitet werden und fehlerhafte Eingaben neu eingefordert werden können.

19.6.5.1 Prüfung der Syntax von Zeichenketten

Stellen Sie sich vor, Sie hätten in einem Gambas-Projekt in einem Formular neben dem Vor- und Nachnamen auch in der postalischen Anschrift die *Postleitzahl*, die *EMail-Adresse* sowie das *Geburtsdatum* und einen *Geldwert* zu erfassen respektive zu prüfen. Nach den bisherigen Kenntnissen ahnen Sie bereits, dass Sie wohl nicht um den Einsatz von regulären Ausdrücken herum kommen, wenn Sie zum Beispiel an die Prüfung einer syntaktisch richtigen EMail-Adresse denken. Den spontanen Gedanken, für die Prüfung Zeichenketten-Funktionen einzusetzen, haben Sie bei der Vielzahl an unterschiedlichen, jedoch syntaktisch korrekten EMail-Adressen erst einmal fallen gelassen.

Als einfachsten Fall suchen Sie sich daher die Postleitzahl heraus. Nichts leichter als das! Die deutsche Postleitzahl besteht aus genau 5 Ziffern und das lässt sich flink auf den folgenden regulären Ausdruck abbilden:

```
Pattern = "^([0-9]){5}$"
```

Ein Testprogramm mit der o.a. Funktion *Match(Subject As String, Pattern As String)* ist schnell entworfen und der Test schien erfolgreich – bis Ihnen jemand Folgendes mitteilte:

- Eine Post-Leitregion in Deutschland umfasst alle Postleitzahlen mit *zwei bestimmten* Ziffern aus dem Bereich 00...99 am Anfang der Postleitzahl.
- Zu den Ziffernkombinationen 00, 05, 43, 62 ist keine Post-Leitregion definiert!

Hätten Sie es gewusst? Abhilfe schafft ein neu formulierter regulärer Ausdruck:

```
"^((?:0[1-46-9]\\d{3})|(?:[1-357-9]\\d{4})|(?:[4][0-24-9]\\d{3})|(?:[6][013-9]\\d{3}))$"
```

Daran hatten Sie ganz bestimmt auch schon gedacht … .

19.6.5.2 Projekt: Prüfung der Syntax von Zeichenketten

In diesem Projekt finden Sie Anregungen, wie man die *Syntax* von ausgewählten Zeichenketten mit klar definierter Semantik durch die Verwendung von regulären Ausdrücken prüfen kann. Bedenken Sie dabei, dass der Einsatz regulärer Ausdrücke ein sehr effektiver Ansatz sein kann. Aber nur dann, wenn das entworfene Muster die zu prüfende Klasse sehr gut und hinreichend fehlerfrei beschreibt! Die im Projekt eingesetzten Muster sind intensiv geprüft worden. Das schließt nicht aus, das es doch noch Einsatzfälle gibt, die nicht korrekt oder für alle Anwendungen abgedeckt werden.

Abbildung 19.6.5.2.1: Verwendung regulärer Ausdrücke (Syntax-Prüfung)

Das vollständige Projekt finden Sie im Download-Bereich. Deshalb entdecken Sie hier nur einen Aus-

zug aus dem Quelltext zur Überprüfung der Syntax einer ISBN 10 – einer internationalen Standard-buchnummer (**I**nternational **S**tandard **B**ook **N**umber). Seit 2007 werden nur noch die ISBN 13 vergeben. Während man die *Syntax* mit einem regulären Ausdruck prüfen kann, muss die korrekte *Prüfziffer* konventionell berechnet werden, weil das mit einem regulären Ausdruck nicht zu leisten ist:

```
Public Sub btnPruefungISBNNummer10_Click()

  sSubject = txtISBNNummer10.Text
  sPattern = "^ISBN\\s(?=[-0-9xX ]{13}$)(?:[0-9]+[- ]){3}[0-9]*[xX0-9]$" ' ISBN 10

  If txtISBNNummer10.Text = "" Then
     Message.Warning("Geben Sie eine ISBN (10) ein!")
     txtISBNNummer10.SetFocus
     Return
  Endif ' txtISBNNummer10.Text = "" ?

  If Match(sSubject, sPattern) = True Then
     SetLEDColor(pbISBN10, "green")
     bISBN_10 = True
     btnISBN_PZ10.Enabled = True
  Else
     SetLEDColor(pbISBN10, "red")
  Endif ' Match(sSubject, sPattern) = True ?

End ' btnPruefungISBNNummer10_Click()
```

Berechnung der Prüfziffer als letztem Zeichen in einer ISBN (10) erfolgt nach einem definierten Algorithmus:

```
Public Sub btnISBN_PZ10_Click()
  Dim iSumme, iPruefZiffer, iPruefZifferISBN, iCount As Integer
  Dim sISBN, iPruefZifferStringISBN As String ' Datentyp String wegen Prüfziffer 10=X

  sISBN = Replace(txtISBNNummer10.Text, "ISBN ", "")
  sISBN = Replace(sISBN, " ", "")
  sISBN = Replace(sISBN, "-", "")
  iSumme = 0
  iPruefZifferStringISBN = Right(sISBN)

  If Upper(iPruefZifferStringISBN) = "X" Then
     iPruefZifferISBN = 10
  Else
     iPruefZifferISBN = Val(iPruefZifferStringISBN)
  Endif ' Upper(iPruefZifferStringISBN) = "X"

  For iCount = 1 To Len(sISBN) - 1
      iSumme = isumme + iCount * Mid(sISBN, icount, 1)
  Next ' iCount

  iPruefZiffer = iSumme Mod 11

  If iPruefZiffer <> iPruefZifferISBN Then
     Message.Error("Fehler!" & Chr(10) & "Die Syntax der ISBN-Nummer (10) ist korrekt." & \
                   Chr(10) & "Die Prüfziffer (ISBN_10) ist jedoch falsch.")
  Else
     Message.Info("Die Prüfziffer ISBN-10-Nummer ist korrekt.")
     btnISBN_PZ10.Enabled = False
  Endif ' iPruefZiffer <> iPruefZifferISBN ?

End ' btnISBN_PZ10_Click()
```

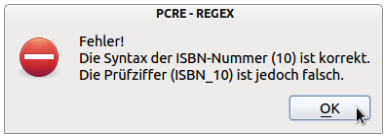

Abbildung 19.6.5.2.2: Kommentierte Berechnung der Prüfziffer einer ISBN (10)