16.13 ComboBox

Die Komponente ComboBox (gb.qt4) ist eine Kombination aus einer *Textbox* und einer *(PopUp-)ListBox*. Das Einsatzfeld einer ComboBox ist die Eingabe von Daten als Zeichenkette. Entweder wählen Sie ein Element aus der Listbox aus, das dann in der Textbox angezeigt wird oder Sie geben Text in das Textfeld ein.

Nur wenn die Eigenschaft *ComboBox.ReadOnly* auf True gesetzt wurde, werden aus den Eingaben valide Daten, mit denen Sie im Programm weiter arbeiten können. Sonst sind die Daten auf jeden Fall zu prüfen!

16.13.1 Eigenschaften und Methoden ComboBox

Ausgewählte Eigenschaften und Methoden einer ComboBox werden jeweils in einer Tabelle aufgeführt und beschrieben:

Eigenschaft	Typ	Beschreibung
Count	Integer	Gibt die Anzahl der Elemente in der Popup-Listbox an.
Current	ComboBox.Item	Liefert das aktuelle, ausgewählte Element in der Popup-Listbox. Combobox.Item repräsentiert einen Eintrag in der Popup-Listbox mit der Eigenschaft Text.
Index	Integer	Gibt den Index des aktuell ausgewählten Elements in der Popup-Listbox an.
Length	Integer	Ermittelt die Länge des ausgewählten Textes in der TextBox.
List	Array	Speichert den Inhalt der Listbox in einem String-Array oder füllt die Listbox mit dem Inhalt eines String-Array.
Maxlength	Integer	Ermittelt die maximal zulässige Länge des Textes im Textfeld oder legt die maximal zulässige Länge des Textes im Textfeld fest.
Password	Boolean	Zeigt an, ob Zeichen in das Textfeld versteckt eingegeben werden und durch Sternchen ersetzt werden.
Pos	Integer	Ermittelt die Position des Cursors im Textfeld oder legt die Position des Cursors im Textfeld fest.
ReadOnly	Boolean	Zeigt an, ob der Inhalt der ComboBox nur gelesen wird oder ob das Textfeld editierbar ist oder nicht.
Sorted	Boolean	Legt fest, ob die Elemente in der Listbox sortiert angezeigt werden oder nicht.
Text	String	Gibt den angezeigten Text in der Textbox zurück oder schreibt den Text in die Textbox. (Synonym: ComboBox.Item.Text)
Selected	Boolean	Gibt an, ob ein Text in der ComboBox ausgewählt ist oder nicht.
Selection	TextBox.Selection	Gibt ein Objekt für die Verwaltung des ausgewählten Text zurück.

Tabelle 16.13.1.1: Übersicht zu Eigenschaften der Komponente ComboBox

TextBox.Selection hat die folgenden Eigenschaften:

- TextBox.Selection.*Text*: Gibt den ausgewählten Text oder Null zurück, wenn *kein* Text markiert ist .
- TextBox.Selection.*Start*: Gibt die Anfangsposition des ausgewählten Textes an oder eine *-1* zurück, wenn *kein* Text markiert ist .
- TextBox.Selection.*Length*: Gibt die Länge des aktuell ausgewählten Textes an oder eine *0* zurück, wenn *kein* Text markiert ist.

Die Methode TextBox.Selection.*Hide* blendet den markierten Text in der TextBox aus.

Die TextBox einer ComboBox verfügt wie jede TextBox über ein Standard-Kontext-Menü, das Sie über einen Klick mit der rechten Maustaste aufrufen.

Eine kurze Beschreibung der Methoden der Komponente C*omboBox* finden Sie in der folgenden Tabelle:

Methode	Beschreibung
.Add(Item As String [,Index As Integer])	Fügt ein Element (Item) in die ComboBox-Liste ein. Ist der optionale Wert *Index* gesetzt, dann erfolgt das Einfügen an der durch den Index definierten Position, sonst am Ende der Liste.
.Insert(Text as String)	Fügt eine Zeichenkette in die leere TextBox ein oder ergänzt den bestehenden Text in der TextBox.
.Clear()	Löscht den Inhalt der ComboBox-Liste.
.Find(Item As String)	Findet ein Element in der Popup-ListBox und gibt den entsprechenden Index (Typ Integer) zurück oder eine -1, wenn das Element nicht in der Popup-ListBox gefunden wird.
.Popup()	Öffnet das ComboBox-Popup-Menü.
.Remove(Index As Integer)	Löscht das indizierte Element in der ComboBox-Liste.
.Select ([Start As Integer, Length As Integer])	Gibt den Text ab der Startposition und in der angegebenen Länge zurück. Ohne (optionale) Argumente wird der gesamte Text zurückgegeben.
.SelectAll()	Selektiert den gesamten Text und gibt ihn zurück.
.Unselect()	Hebt die Auswahl des markierten Textes in der ComboBox auf.

Tabelle 16.13.1.2: Übersicht zu Methoden der Klasse ComboBox

16.13.2 Ereignisse ComboBox

Die speziellen Ereignisse der Komponente C*omboBox* und ergänzende Kommentare finden Sie hier:

Ereignis	Beschreibung
Activate	Wird ausgelöst, wenn die ComboBox den Fokus besitzt und die Enter-Taste gedrückt wurde.
Change	Wird ausgelöst, wenn der Text in der Combobox geändert wird und die Eigenschaft ComboBox.ReadOnly auf False gesetzt ist.
Click	Wird ausgelöst, wenn ein Element in der Popup-Liste ausgewählt wird.

Tabelle 16.13.2.1: Übersicht zu den wichtigsten Ereignissen der Komponente ComboBox

Das Click-Ereignis wird ausgelöst, wenn

- der User einen Eintrag in der Liste durch einen Klick (LMT) ausgewählt hat oder
- die Index-Eigenschaft gesetzt wurde oder
- die Text-Eigenschaft gesetzt wurde.

Hinweise zum Ereignis ComboBox_Click()

- Das Click-Ereignis wird ausgelöst, wenn die Index- oder Text-Eigenschaft festgelegt wurde, auch wenn der neue Wert der gleiche wie der alte ist! Das gilt nicht beim Einsatz der Komponente *gb.qt4*.
- Das Click-Ereignis wird nicht ausgelöst, wenn Sie den Index auf *-1* setzen oder wenn Sie die Text-Eigenschaft mit etwas füllen, das *nicht* bereits in ComboBox-Liste steht.
- Das Click-Ereignis wird nicht ausgelöst, wenn der Inhalt der Liste durch Hinzufügen, Entfernen oder Löschen geändert wurde oder Sie die List-Eigenschaft ändern.

16.13.3 ComboBox-Listen

Bevor Sie eine ComboBox für die Eingabe von Daten (Datentyp Zeichenkette) in einem Programm nutzen können, müssen Sie die Liste (Datentyp Array) füllen, aus denen der Benutzer eine Auswahl treffen kann. Welchen der vorgestellten Ansätze Sie favorisieren hängt von den zu lösenden Aufgaben und dem Wert der Eigenschaft *ComboBox.ReadOnly* ab. Zuerst werden einzelne Ansätze vorgestellt und anschließend mit Beispielen umgesetzt:

- Einfügen der Elemente in die Liste (Inline-)Array,
- Einfügen der Elemente in die Liste mit der Add-Methode,
- Einfügen der Elemente in ein explizit deklariertes Array, das später als Listeninhalt übergeben wird.

Import der Elemente einer Liste zum Beispiel

- aus einer binären, gambas-spezifischen Datei,
- aus einer XML-Datei, csv-Datei oder einer einfachen Text-Datei,
- aus einer Datenbank-Tabelle (Filter setzen in SQL-Anweisung für genau ein Feld)
- aus programm-internen Berechnungen (→ Fallunterscheidungen),
- aus einer generierten Schnittstellen-Liste oder
- aus einer generierten Liste der im System verfügbaren Drucker.

16.13.4 Beispiel 1 – ComboBox-Liste füllen

```
[1] ' Gambas class file
[2]
[3] Public Sub _new()
[4]   SetCBList()
[5] End ' _new()
[6]
[7] Public Sub Form_Open()
[8]   FMain.Center
[9]   FMain.Resizable = False
[10]End ' Form_Open()
[11]
[12]Public Sub SetCBList()
[13]  Dim aMatrix As New String[]
[14]
[15]  ComboBox1.List = ["Element2", "Element1"] ' Inline-Array
[16]  ComboBox1.Add("Element3")
[17]  ComboBox1.Add("Element5")
[18]  ComboBox1.Add("Element4")
[19]  aMatrix.Add("Element6")
[20]  aMatrix.Add("Element7")
[21]  ComboBox1.List = ComboBox1.List.Insert(aMatrix)
[22]  ComboBox1.Sorted = True
[23]  ComboBox1.Text = ComboBox1.List[0]
[24]
[25]End ' SetList()
```

Den Aufruf von *SetCBList()* sollten Sie in die _new-Prozedur auslagern, weil die Anweisungen

```
ComboBox1.Text = ComboBox1.List[0] ' Anzeige erster Eintrag in der CB-Liste
ComboBox1.Text = ComboBox1.List[ComboBox1.List.Max] ' Anzeige letzter Eintrag in der CB-Liste
```

das Klick-Ereignis auslösen würden!

Kommentare:

- Die Liste wird in Zeile 15 mit 2 Elementen gefüllt (Inline-Array).
- Die Liste wird um 3 Einträge erweitert.
- In den Zeilen 19 und 20 werden 2 Elemente in das Array aMatrix eingefügt.
- Der Inhalt des Arrays aMatrix wird in der Zeile 21 *an das Ende* der Liste eingefügt.
- Anschließend werden die Einträge in der Liste sortiert.

16.13.5 Beispiel 2 – ComboBox mit History

In diesem Beispiel wird einer ComboBox eine History spendiert. Damit werden hier die letzten Pfad-Einträge beim Programmstart in die Liste der ComboBox eingelesen. Neuere Pfade werden zur Laufzeit *dynamisch* in die Liste eingefügt und überschreiben ältere Einträge. Der Quelltext zeigt, dass der aktuelle Listeninhalt in einer Konfigurationsdatei gespeichert wird, weil die Komponente *gb.settings* benutzt wird, die alle Dateioperationen für das Exportieren und Importieren der Listeninhalte implizit bereitstellt.

```
' Gambas class file

Private $aFavorites As String[]
Private Const $iCount As Integer = 4

Public Sub Form_Open()

  FMain.Center
  $aFavorites = Settings["Favorites"] ' Import
' Konfigurationsdatei: /home/hans/.config/gambas3/ComboBoxFavourites.conf
  If Not $aFavorites Then $aFavorites = New String[]
  ReloadFavorites()

End ' Form_Open()

Public Sub btnSelect_Click()
' Nur existierende Verzeichnisse erfassen

  If Not IsDir(cmbPath.Text) Then
    Message.Error(("Der angegebene Pfad ist kein Verzeichnis!"))
    cmbPath.SetFocus
    Return
  Endif ' Not IsDir(cmbPath.Text) ?

' Keine Duplikate zulassen und History begrenzen
  If $aFavorites.Count <= $iCount Then
     If Not $aFavorites.Exist(cmbPath.Text) Then $aFavorites.Add(cmbPath.Text)
  Else
     If Not $aFavorites.Exist(cmbPath.Text) Then
        $aFavorites.Remove(0)
        $aFavorites.Add(cmbPath.Text)
     Endif
  Endif
  ReloadFavorites()

End ' btnSelect_Click()

Public Sub Form_Close()
  Settings["Favorites"] = $aFavorites ' Export
End ' Form_Close()

'****************************************************************************************

Private Sub ReloadFavorites()
' Ein leerer Eintrag für die Eingabe in der TextBox, dann die Favoriten
  cmbPath.List = [""].Insert($aFavorites)
End ' ReloadFavorites()

Public Sub dchPath_Change()
  cmbPath.Text = dchPath.Value
End ' dchPath_Change()

Public Sub cmbPath_Activate()
  btnSelect_Click()
End ' cmbPath_Activate()
```

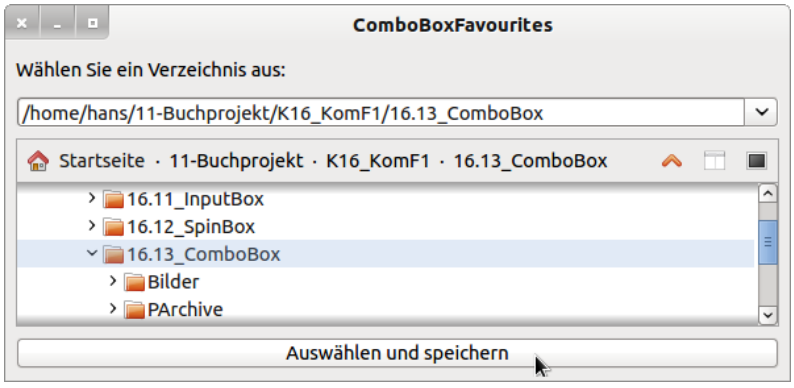

Abbildung 16.13.5.1: ComboBox mit History

16.13.6 Beispiel 3 – ComboBoxen

Bei der Entwicklung eines Programms zur Messung von Temperaturen konnte eine Platine von Stephan Mischnik (www.strippenstrolch.de) eingesetzt werden. Es war nur bekannt, dass als Temperatur-Sensor ein NTC dient, ein PIC 08M2+ als AD-Wandler eingesetzt wurde und die Temperatur-Daten über eine RS232-Schnittstelle eingelesen werden könnten. Um den Programmtest offen zu halten, wurden für den RS232-Port eine Combobox und für die Einstellung der einzelnen Übertragungsparameter der seriellen Schnittstelle 5 Comboboxen verwendet, deren *.ReadOnly*-Eigenschaft jeweils auf True gesetzt wurden:

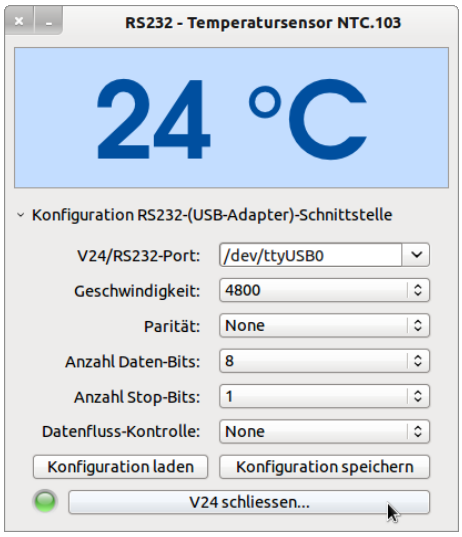

Abbildung 16.13.6.1: Konfiguration RS232

Hier interessiert nur der Einsatz der 6 ComboBoxen. Daher werden nur ausgewählte Passagen aus dem Quelltext vorgestellt. Die Auszüge zeigen, wie die Listen der ComboBoxen *cmbSpeed, cmbParity, cmbDataBits, cmbStopBits* und *cmbFlow* statisch – bezogen auf Anzahl und Werte – gefüllt werden. Eine Besonderheit besteht darin, dass die Liste der ComboBox *cmbRs232PortName* erst zur Laufzeit in Abhängigkeit von den auf dem System gefundenen seriellen Schnittstellen oder den USB-RS232-Adapter-Schnittstellen *dynamisch* gefüllt wird:

```
[1] Public Sub Form_Open()
[2]   Dim aDataFlow As New String[]
[3] …
[4]   aDataFlow.Add("None")
[5]   aDataFlow.Add("XON/XOFF")
[6]   aDataFlow.Add("RFR/CTS")
[7]   aDataFlow.Add("RFR/CTS + XON/XOFF")
[8]   cmbSpeed.List = ["2400", "4800", "9600"]
[9]   cmbParity.Add("None")
[10]  cmbParity.Add("Even")
[11]  cmbParity.Add("Odd")
[12]  cmbDataBits.List = ["5", "6", "7", "8"]
[13]  cmbStopBits.List = ["1", "2"]
[14]  cmbFlow.List = aDataFlow
[15]
[16]  RS232PortListeGenerieren()
[17]  …
[18]End ' Form_Open
[19]
[20]Public Sub RS232PortListeGenerieren()
[21]  Dim iCount As Integer
[22]  Dim sZeile, sListeV24, sListeUSB, s As String
[23]  Dim aSchnittstellenMatrix As New String[]
[24]  Dim aListe As New String[]
[25]  Dim hProcess As Process
[26]
[27]  cmbRS232PortName.Clear()
[28]
[29]' Ermittlung echter RS232-Schnittstellen
[30]  Shell "dmesg | grep ttyS | grep 00:" To sListeV24
```

```
[31]  If Len(sListeV24) > 0 Then
[32]     aSchnittstellenMatrix = Split(sListeV24, " ")
[33]     For Each sZeile In aSchnittstellenMatrix
[34]         If InStr(sZeile, "ttyS") Then
[35]            cmbRS232PortName.Add("/dev/" & Trim$(sZeile))
[36]         Endif
[37]     Next ' FOR EACH
[38]  Endif ' Len(sListeV24) > 0 ?
[39]
[40]' Ermittlung USB-RS232-Adapter-Schnittstellen
[41]  Shell "dmesg | grep ttyUSB" To sListeUSB
[42]  If Len(sListeUSB) > 0 Then
[43]     aSchnittstellenMatrix = Split(sListeUSB, "\n")
[44]     For Each sZeile In aSchnittstellenMatrix
[45]         For iCount = 0 To 7
[46]             If InStr(sZeile, "ttyUSB" & CInt(iCount)) Then
[47]                aListe.Add("ttyUSB" & CInt(iCount))
[48]             Endif
[49]         Next ' iCount
[50]     Next ' FOR EACH
[51]  Endif ' Len(sListeUSB) > 0 ?
[52]
[53]  aListe.Sort
[54]  aListe = RemoveMultiple(aListe)
[55]
[56]  For iCount = 0 To aListe.Max
[57]      cmbRS232PortName.Add("/dev/" & Trim$(aListe[iCount]))
[58]  Next ' iCount
[59]
[60]  If cmbRS232PortName.Count = 0
[61]     cmbRS232PortName.Background = Color.RGB(255, 191, 191)
[62]     cmbRS232PortName.Add("           Keine RS232-Schnittstelle gefunden!")
[63]     btnOnOff.Enabled = False
[64]  Endif ' cmbRS232PortName.Count = 0 ?
[65]
[66]End ' RS232PortListeGenerieren()
[67]
[68]Public Function RemoveMultiple(aStringListe As String[]) As String[]
[69]  Dim iCount As Integer
[70]  Dim iIndex As Integer
[71]  Dim sElement As String
[72]
[73]  iIndex = 0 ' Initialisierung NICHT notwendig
[74]  While iIndex < aStringListe.Count
[75]    iCount = 0
[76]    sElement = aStringListe[iIndex]
[77]    While aStringListe.Find(sElement) <> -1
[78]      Inc iCount
[79]      aStringListe.Remove(aStringListe.Find(sElement))
[80]    Wend
[81]    If iCount Mod 2 = 1 Then
[82]       aStringListe.Add(sElement, iIndex)
[83]       Inc iIndex
[84]    Endif ' iCount Mod 2 = 1 ?
[85]  Wend
[86]
[87]  Return aStringListe
[88]
[89]End ' RemoveMultiple(...)
```

Nach dem Einstellen der richtigen Übertragungsparameter konnte der Expander eingeklappt werden:

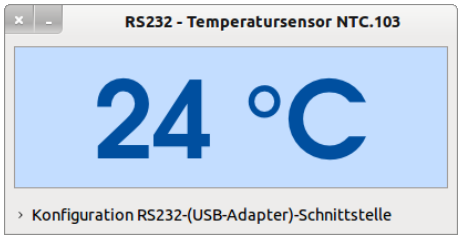

Abbildung 16.13.6.2: Temperaturmessung

16.13.7 Beispiel 4 – ComboBoxen zur Laufzeit bearbeiten (Export - Import)

Wenn Sie die Eigenschaft *.ReadOnly* von Comboboxen auf True setzen, dann sichern Sie, dass nur valide Daten aus diesen ComboBoxen im Programm verarbeitet werden. Die Listen können Sie dann aber nicht mehr *direkt* in der ComboBox ändern, was aber gelegentlich erforderlich wäre. Im folgenden Projekt wird die Möglichkeit beschrieben, wie Sie die Listen ausgewählter ComboBoxen zur Laufzeit des Programms bearbeiten, exportieren und importieren.

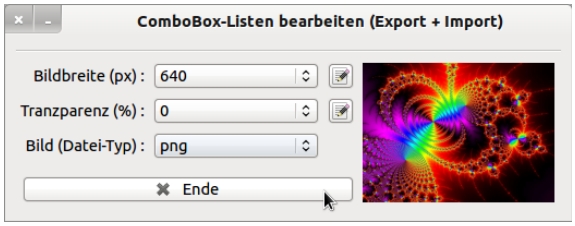

Abbildung 16.13.7.1: Drei ComboBoxen

Beim *ersten* Programmstart werden die Listen für alle drei ComboBoxen mit zweckbestimmten Werten gefüllt. Über einen Klick auf den Button rechts neben der ComboBox wird die Bearbeitung der entsprechenden Liste angeschoben. Wenn Sie auf einen (zusätzlichen) Button verzichten wollen, dann starten Sie Bearbeitung einfach über das zugewiesene Kontext-Menü, über das die ComboBox 3 (Datei-Typ) verfügt:

Abbildung 16.13.7.2: Bearbeitung von ComboBox-Listen (Laufzeit)

Vor dem Schließen des Programmfensters werden die Inhalte der Listen der 3 ComboBoxen in einer jeweils eigenen Datei im Projekt-Verzeichnis gespeichert (Listen-Export). Bei jedem weiteren Neustart des Haupt-Programms werden die Inhalte der drei Dateien separat ausgelesen und den Listen der drei ComboBoxen zugewiesen (Listen-Import).

Der recht umfangreiche, aber gut dokumentierte Quelltext des Projekts wird hier nicht angegeben. Es lohnt auf jeden Fall, sich den kompletten Quelltext genau anzusehen. Die Quelltexte werden Ihnen viel Neues offerieren und Ihre Sichtweise auf die Programmierung mit Gambas verändern. Das liegt vor allem daran, dass der Quelltext für die Bearbeitung von Listen einen Ausschnitt aus dem Quelltext der Gambas-IDE repräsentiert.