19.1.2 Beispiele – Methoden Write und Read

Die vorgestellten Projekte sind erprobt. Die Programme werden für einen Langzeitversuch zur Erfassung von Temperaturen in unterschiedlichen Höhen in einem Mikroklima genutzt.
Die beiden Methoden *Write* und *Read* zum direkten Schreiben in eine Konfigurationsdatei sowie zum direkten Lesen werden am Beispiel eines Programmfensters zum Abschluss dieses Kapitels behandelt.

19.1.2.1 Beispiel 1 – Projekt Temperaturmessung

Das Projekt *Temperaturmessung* verfolgt die Idee, die analoge Größe Temperatur mit einem NTC-Temperatursensor zu erfassen und mit einem Analog-Digital-Wandler in einen digitalen Datenstrom umzuwandeln, der über eine serielle RS232-Schnittstelle (V24-Schnittstelle) respektive über einen RS232-USB-Adapter eingelesen wird, um diesen Datenstrom dann auszuwerten und als Temperaturwert anzuzeigen:

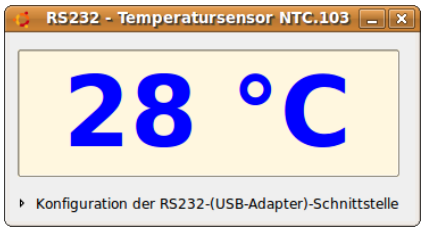

Abbildung 19.1.2.1.1: Anzeige des Temperaturwertes

Die serielle Schnittstelle ist dabei auf die *Übertragungsparameter* einzustellen, die eine fehlerfreie Übertragung der Signale sichern. Während der Erprobung sollten verschiedene Einstellungen getestet werden, die im produktiven Einsatz der Platine als optimale Vorgabewerte bei jeder weiteren Messung einzulesen sind. Es ist deshalb günstig, über den ausgerollten Expander ► sofort Zugriff auf die Konfiguration der RS232-Schnittstelle zu haben:

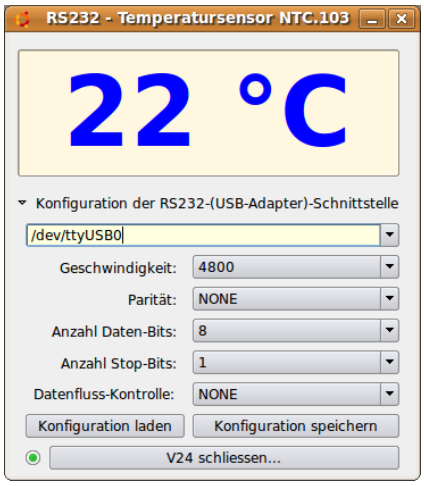

Abbildung 19.1.2.1.2: Einstellung der Übertragungsparameter

Daher wird auch der Status der Komponente *Expander* in der Konfigurationsdatei erfasst. Die technischen Details zur Schnittstelle und zur Platine werden hier nicht weiter besprochen. Sie erfahren aber alles, wie man mit der Komponente *Settings* und Konfigurationsdateien arbeitet. *Settings* verfügt nur über wenige Eigenschaften und Methoden, von denen vorrangig folgende genutzt werden:

Methode	Beschreibung
`Settings.Reload`	Diese Methode ändert alle gesetzten Konfigurationseinstellungen und überschreibt diese durch die Werte in der Konfigurationsdatei.
`Settings.Save`	Beim Aufruf dieser Methode werden alle explizit gesetzten Konfigurationseinstellungen in der Konfigurationsdatei gespeichert und überschreiben die vorhandenen Werte. Alle anderen Werte bleiben erhalten.

	Das Speichern der aktuellen Konfigurationseinstellungen erfolgt *automatisch*, wenn das Programm beendet wird.

Tabelle: 19.1.2.1 Beschreibung ausgewählter Methoden

Wenn Sie den *Standard-Pfad für die Konfigurationsdatei* nutzen, wird die Konfigurationsdatei automatisch angelegt. Auch das Speichern der Konfigurationsdaten in der Konfigurationsdatei beim Beenden des Programms und das Öffnen der Konfigurationsdatei zum Einlesen der Konfigurationsdaten werden vom Programm übernommen. Es besteht die Möglichkeit, zur Laufzeit eines Programms die aktuellen Konfigurationsdaten zwischenzuspeichern oder den Inhalt einer Konfigurationsdatei neu einzulesen.

Eigenschaft	Beschreibung
Settings.Path	Diese Eigenschaft gibt das Verzeichnis an, in dem die Konfigurationsdatei gespeichert wird oder setzt das Verzeichnis.

Tabelle: 19.1.2.2 Beschreibung der Eigenschaften

Der unten aufgeführte Inhalt (Auszug) der Konfigurationsdatei für das Projekt RS232T wurde vom Programm generiert. Entweder geschieht das Speichen beim Beenden des Programms oder während der Programmausführung (manuell oder in Abhängigkeit von bestimmten Programmzuständen) mit der Methode *Settings.Save*. Beachten Sie, dass die 2 Sektionen alphabetisch geordnet eingetragen werden – unabhängig von der Notation im Programm-Quelltext:

```
[Expander]
ExpanderHiddenStatus=True

[V24Konfiguration]
Port-Name="/dev/ttyUSB0"
Geschwindigkeit="4800"
...
AnzahlStopbits="1"
Datenflusskontrolle="NONE"
```

Anschließend werden wesentliche Teile des Quelltextes angegeben und anschließend erläutert, weil nur dann das Zusammenspiel zwischen den einzelnen Prozeduren und dem Einsatz der Konfigurationsdatei deutlich wird:

```
' Gambas class file

PUBLIC v24Settings AS Settings
PUBLIC iTemperaturByte AS Byte

PUBLIC SUB Form_Open()
  FMain.Center
  FMain.Border = 1
  FMain.Height = 166
  expRX_TX.Hidden = TRUE
  expRX_TX.Animated = TRUE

  v24Settings = NEW Settings

  SetConfigurationValues()
  expRX_TX.Hidden = v24Settings["Expander/ExpanderHiddenStatus", "TRUE"]
  ...
END ' Form_Open

PUBLIC SUB Form_Show()
  RS232ListeGenerieren
END ' Form_Show

...

PUBLIC SUB btnKonfigurationNeuEinlesen_Click()
  v24Settings.Reload
  SetConfigurationValues()
END ' KonfigurationNeuEinlesen

PUBLIC SUB btnAktuelleKonfigurationSpeichern_Click()
  IF Message.Question("Soll die aktuelle RS232-Konfiguration gespeichert werden?", "Ja", "Nein") = 1 THEN
     GetConfigurationValues()
     v24Settings["Expander/ExpanderHiddenStatus"] = expRX_TX.Hidden
     expRX_TX.Hidden = FALSE
     v24Settings.Save
  ELSE
     RETURN
```

```
  ENDIF ' Message.Question("...?")
END ' AktuelleKonfigurationSpeichern?

PUBLIC SUB SetConfigurationValues()

  cmbRS232PortName.Text = v24Settings["V24Konfiguration/Port-Name", "/dev/ttyS0"]
  cmbSpeed.Text = v24Settings["V24Konfiguration/Geschwindigkeit", "4800"]
  cmbParity.Text = v24Settings["V24Konfiguration/Parität", "NONE"]
  cmbDataBits.Text = v24Settings["V24Konfiguration/AnzahlDatenbits", "8"]
  cmbStopBits.Text = v24Settings["V24Konfiguration/AnzahlStopbits", "1"]
  cmbFlow.Text = v24Settings["V24Konfiguration/Datenflusskontrolle", "NONE"]

END ' SetConfigurationValues()

PUBLIC SUB GetConfigurationValues()

  v24Settings["V24Konfiguration/Port-Name"] = cmbRS232PortName.Text
  v24Settings["V24Konfiguration/Geschwindigkeit"] = cmbSpeed.Text
  v24Settings["V24Konfiguration/Parität"] = cmbParity.Text
  v24Settings["V24Konfiguration/AnzahlDatenbits"] = cmbDataBits.Text
  v24Settings["V24Konfiguration/AnzahlStopbits"] = cmbStopBits.Text
  v24Settings["V24Konfiguration/Datenflusskontrolle"] = cmbFlow.Text

END ' GetConfigurationValues()

PUBLIC SUB Form_Close()
  IF RS232.Status = Net.Active THEN CLOSE RS232
  GetConfigurationValues()
  v24Settings["Expander/ExpanderHiddenStatus"] = expRX_TX.Hidden
' V24Settings.Save wird bei Form_Close automatisch ausgelöst
END ' Form_Close
```

Den vollständigen Quelltext zum erprobten Projekt *RS232T* finden Sie im Download-Bereich. Erläuterungen zu den technischen Details des Datentransfers über eine serielle Schnittstelle entdecken Sie dagegen im Kapitel 24.10.

Hinweise zum Quelltext

Es wurde festgelegt, für die Konfigurationsdateien den Standard-Pfad zu nutzen

```
  v24Settings = NEW Settings
```

und es wird beim Programmstart eine neue Instanz der Klasse gb.settings angelegt. Die Abbildung *19.1.1.2* zeigt die Übertragungsparameter der V24-Schnittstelle, deren Werte durch die Angaben in der Konfigurationsdatei beim Programmstart bestimmt werden. Beim *ersten Programmstart* wird im Standardpfad nach der Konfigurationsdatei gesucht, die den Namen des Projektes trägt und die Extension 'conf' hat:

```
User.Home &/ ".config/gambas" &/ RS232T & ".conf"
```

Da diese Datei *nicht* vorhanden ist – also keine Schlüssel-Wert-Paare existieren – werden die notierten *Vorgabewerte* im Programm zugewiesen. Daher startet das Programm *RS232T* zum Beispiel mit den (vor-)eingestellten 8 Datenbits sowie mit minimiertem Expander:

```
  SetConfigurationValues()
  expRX_TX.Hidden = v24Settings["Expander/ExpanderHiddenStatus", "TRUE"]
```

Die weiteren Vorgabewerte sind erprobt oder durch den AD-Wandler zum Teil vorgegeben. Sie können diese Werte im Rahmen der Gültigkeitsgrenzen für die Übertragungsparameter einer RS232-Schnittstelle nach folgender Syntax selbst festlegen:

```
Objektname.Eigenschaftswert = SettingObjektname["Sektionsname/SchlüsselName", optional "Vorgabewert"]
```

Das Programm liest diese Vorgabewerte beim *ersten* Programmstart oder bei *fehlender* Konfigurationsdatei automatisch aus und setzt die entsprechenden Werte der ausgewählten Eigenschaften direkt oder über den aktuellen Wert, der in der ComboBox angezeigt wird.

Bei der Arbeit mit dem Programm *RS232T* können Sie die Werte zur Einstellung der Übertragungsparameter der verwendeten seriellen Schnittstelle so ändern und erproben, um zum Beispiel beim Datenaustausch zwischen Ihrem Computer und einem angeschlossenen Temperatursensor mit Analog-Digital-Wandler ein Optimum anzustreben.

Beenden Sie das Programm, werden ausgewählte Werte ganz bestimmter Objekte oder Variablen in der Konfigurationsdatei *automatisch* gespeichert und stehen beim nächsten Programmstart als Vorgabewerte zur Verfügung:

```
PUBLIC SUB Form_Close()
  IF RS232.Status = Net.Active THEN CLOSE RS232
  GetConfigurationValues()
  v24Settings["Expander/ExpanderHiddenStatus"] = expRX_TX.Hidden
' V24Settings.Save wird bei Form_Close automatisch ausgelöst
END ' Form_Close
```

Wenn Sie zur Laufzeit des Programms *RS232T* verschiedene Einstellungen der Übertragungsparameter erproben wollen, dann können Sie zum Beispiel eine *fehlerfreie Konfiguration* der Parameter zwischenspeichern. Das Speichern erfolgt in diesem Fall nicht automatisch, sondern mit der Anweisung *v24Settings.Save*:

```
PUBLIC SUB btnAktuelleKonfigurationSpeichern_Click()
  IF Message.Question("Soll die aktuelle RS232-Konfiguration gespeichert werden?", "Ja", "Nein") = 1 THEN
     GetConfigurationValues()
     v24Settings["Expander/ExpanderHiddenStatus"] = expRX_TX.Hidden
     expRX_TX.Hidden = FALSE
     v24Settings.Save
  ELSE
     RETURN
  ENDIF ' Message.Question("...?")
END ' AktuelleKonfigurationSpeichern?
```

Tritt bei der weiteren Erprobung mit geänderten Übertragungsparametern ein Fehler auf oder eine Verschlechterung der Übertragungsqualität der V24-Schnittstelle, können Sie die in der Konfigurationsdatei zuletzt gespeicherten Werte einlesen, wenn Sie von denen annehmen können, das diese zu einer fehlerfreien Konfiguration der RS232-Schnittstelle gehören. Dabei werden die bisher genutzten Werte der Übertragungsparameter gelöscht und durch die eingelesenen Werte aus der Konfigurationsdatei – hier ohne Rückfrage – überschrieben:

```
PUBLIC SUB btnKonfigurationNeuEinlesen_Click()

  v24Settings.Reload
  SetConfigurationValues()

END ' KonfigurationNeuEinlesen
```

Sie können aber auch ganz gezielt nur *ausgewählte Konfigurationsdaten* in einer Sektion überschreiben. Alle anderen gesetzten Werte in der Konfigurationsdatei werden dadurch *nicht* verändert:

```
PUBLIC SUB btnStandardEinstellungenLaden2_Click()

' Nur für den RS232-Übertragungsparameter Geschwindigkeit den Wert setzen
  v24Settings["V24Konfiguration/Geschwindigkeit"] = "9600"
  v24Settings.Save
  v24Settings.Reload
' Standardwert zuweisen
  cmbSpeed.Text = v24Settings["V24Konfiguration/Geschwindigkeit"]

END ' StandardEinstellungenLaden2
```

Die Möglichkeit wird im Programm RS232TS nicht genutzt.

Achtung:
Weisen Sie einem Schlüssel-Wert-Paar eine leere Zeichenkette oder den Wert NULL zu, dann wird dieses Paar in der entsprechenden Sektion gelöscht. Gibt es dann keine weiteren Schlüssel-Wert-Paare mehr in der Sektion, wird auch die Sektion gelöscht!

19.1.2.2 Beispiel 2 – Projekt Temperaturmessung TS

Immer dann, wenn Sie bestimmte Dateien im Programm nutzen oder Konfigurationsdateien zentral in einem Verzeichnis verwalten wollen, benötigen Sie für ein Programm die Konfigurationsdatei in einem bestimmten Verzeichnis. Für das zweite Projekt Temperaturmessung wird deshalb *nicht* der Standardpfad für die Konfigurationsdatei genutzt, sondern ein eigenes Verzeichnis im Home-Verzeichnis mit frei vergebenen Namen für das Verzeichnis und die Konfigurationsdatei.

Die Änderungen im Quelltext gegenüber Beispiel 1 sind minimal. Deshalb werden nur die Änderungen oder Ergänzungen angegeben. Das Verzeichnis bekommt den Namen *V24TS*. Die Konfigurationsdatei erhält den Dateinamen *v24TS.conf* und ist sichtbar. Sie müssen sich bei der Nutzung eines benutzerdefinierten Pfads selbst um das Anlegen des Ordners *V24TS* und der Konfigurationsdatei *v24TS.conf* kümmern.

Das Verzeichnis *V24TS* und die Konfigurationsdatei *v24TS.conf* werden nur dann angelegt, wenn diese nicht existieren und werden im Normalfall beim ersten Start des Programms angelegt.
Quelltext:

```
' Gambas class file

PUBLIC v24Settings AS Settings
PUBLIC hDatei AS File
PUBLIC iTemperaturByte AS Byte

PUBLIC SUB Form_Open()

  FMain.Center
  FMain.Border = 1
  FMain.Height = 166
  expRX_TX.Hidden = TRUE
  expRX_TX.Animated = TRUE
  rbLED.ForeColor = Color.Red

  IF NOT Exist(User.Home &/ "V24TS") THEN
     MKDIR User.Home &/ "V24TS"
  ENDIF
  IF NOT Exist(User.Home &/ "V24TS" &/ "v24TS.conf") THEN
     hDatei = OPEN (User.Home &/ "V24TS" &/ "v24TS.conf") FOR CREATE
     hDatei.Close
  ENDIF

  v24Settings = NEW Settings(User.Home &/ "V24TS" &/ "v24TS.conf")

  SetConfigurationValues()
  expRX_TX.Hidden = v24Settings["Expander/ExpanderHiddenStatus", "TRUE"]

  cmbRS232PortName.Background = Color.RGB(255, 255, 223)
  Timer1.Start ' Alternative: Timer1.Enabled = TRUE
  Timer1.Delay = 100 ' Alle 100ms wird die Temperatur ausgelesen und angezeigt
  btnOnOff.Text = "V24 öffnen und Temperatur anzeigen."

END ' Form_Open
```

Das ist eine Alternative zum Anlegen der Konfigurationsdatei:

```
  sSettingsPath = User.Home &/ "V24TS/v24TS.conf"
  IF NOT Exist(sSettingsPath) THEN
     TRY File.Save(sSettingsPath, "")
     IF ERROR THEN
        Message.Error("Die Datei " &  File.Name(sSettingsPath) & " konnte nicht angelegt werden!")
        RETURN
     ENDIF ' ERROR
  ENDIF ' NOT Exist()?
```

19.1.2.3 Settings-Methoden Read und Write

Verwenden Sie den folgenden Code in den zwei angegebenen Prozeduren, können Sie auch für die Größe und Lage des Programmfensters geeignete Startwerte setzen oder diese beim Programmende mit den aktuellen Werten (automatisch) speichern:

```
PUBLIC SUB Form_Open()
...
FMain.Top = v24Settings["Window/Top", FMain.Top]
FMain.Left = v24Settings["Window/Left", FMain.Left]
FMain.Height = v24Settings["Window/Height", FMain.Height]
FMain.Width = v24Settings["Window/Width", FMain.Width]
...
End ' Form_Open
```

```
PUBLIC SUB Form_Close()
...
v24Settings["Window/Top"] = FMain.Top
v24Settings["Window/Left"] = FMain.Left
v24Settings["Window/Height"] = FMain.Height
v24Settings["Window/Width"] = FMain.Width
```

```
...
End ' Form_Close
```

Eine Besonderheit besteht darin, dass Sie explizit *keine* Vorgabewerte für die Fenstergröße eintragen müssen. Starten Sie das Programm zum ersten Mal, werden die zur *Entwurfszeit* festgelegten Fenstergrößen (FMain.Top, FMain.Left, FMain.Height, FMain.Width) als Vorgabewerte genutzt. Jedoch nur dann, wenn Sie über die Fenstereigenschaft *FMain.Border = 2 (Resizable)* eine Änderung der Fenstergröße zulassen. Sonst werden die Eigenschaften zur Fenstergröße aus der Konfigurationsdatei ignoriert!

Um die Koordinaten des Programmfensters zur Laufzeit in die Konfigurationsdatei einzufügen oder auszulesen und als Koordinaten des Programmfensters zu setzen, setzen Sie die beiden Methoden *Write* und *Read* ein:

```
PUBLIC SUB Form_Verschieben_Click()

 mySettings.Write(ME, "Form")
   FMain.Move(100, 100)
   WAIT 2
 mySettings.Read(ME, "Form")

END '  Form_Verschieben
```

Die Anweisung

```
 mySettings.Write(ME, "Form")
```

fügt die aktuellen Koordinaten des Programmfensters *direkt* in die Konfigurationsdatei ein, wobei der Name der Sektion (Form) und der Schlüsselname (Geometry) automatisch festgelegt werden:

```
[Form]
Geometry=[577,378,525,294]
```

Nach dem Verschieben des Fensters in die linke obere Bildschirmecke kehrt das Fenster nach 2 Sekunden an die alte Position zurück, wenn die gespeicherten Koordinaten ausgelesen und gesetzt werden:

```
 mySettings.Read(ME, "Form")
```

DOWNLOAD