12.2.0 Form – Klasse Window und Klasse Form

In diesem Kapitel geht es um Beschreibungen zu den Klassen Window und Form sowie um die Besonderheiten von Formularen.

12.2.0.1 Klasse Window und Klasse Form

Die Klasse Form erbt von der Klasse Window. Das Wiki http://gambaswiki.org/wiki/comp/gb.qt4/form listet noch ein paar besondere Eigenschaften von Form gegenüber Window auf. Eines ist bemerkenswert: Die Klasse Form ist auf die Gambas-IDE ausgerichtet und ein Formular in der Gambas-IDE beschreibt immer eine Klasse, die von der Klasse Form erbt. Deshalb dürfen Sie auch in einer .class-Datei zu einem Formular nicht das Inherits-Schlüsselwort benutzen. Ein Formular erbt implizit schon von der Form-Klasse und Gambas unterstützt das gleichzeitige Erben von mehreren Klassen nicht. Jedes Formular wird also zu einer speziellen Form-Klasse. Von dieser Klasse bekommen Sie eine sogenannte *automatische Instanz* – Classes[Formular].Instance – frei Haus, die ein Objekt dieser Klasse ist, das Sie über den Klassennamen ansprechen. Das werden Sie sicher schon mehr oder weniger bewusst festgestellt haben: Sie haben ein Formular 'FMain' und ein echtes Anwendungsfenster als eine Instanz Ihrer Formular-Klasse, das automatisch erscheint und das Sie über den Namen 'FMain' ansprechen können.

Den Quelltext einer Form-Klasse schreiben Sie weitgehend selbst und er steht nach dem Speichern in der .class-Datei, die zum Formular gehört. Der Gambas-Compiler generiert aber noch eine versteckte, spezielle Routine, die '$load' heißt. Diese wird bei der Instantiierung eines Form-Objektes intern aufgerufen und sorgt dafür, dass alle Steuerelemente, die man im IDE-Form-Editor platziert, erzeugt werden. Davon können Sie überzeugen, indem Sie einen Projekt-Ordner öffnen, in dem ein (einfaches) graphisches Projekt mit mindestens einem Formular existiert. Führen Sie dann in der System-Konsole den folgenden Befehl aus:

```
~/PROJEKT-ORDNER/ $ gbc3 -av
```

Der Schalter 'v' veranlasst den Compiler, den Byte-Code des Kompilats menschenlesbar auszugeben. Scrollen Sie durch diese Ausgabe, so werden Sie u.a. die $load-Funktion finden. Sie können erkennen, dass die Funktion alle Steuerelemente auf dem Formular erzeugt und jeweils mit den angegebenen Eigenschaften versieht:

```
Private Sub {$load}()
  With Me
    .MoveScaled(0,0,48,19)

  {SpinBox1} = New SpinBox(Me) As "SpinBox1"
  With {SpinBox1}
    .MoveScaled(2,2,32,4)
    .Value = 50
  End With

  {SpinBar1} = New SpinBar(Me) As "SpinBar1"
  With {SpinBar1}
    .MoveScaled(2,7,32,4)
    .Value = 0.7
    .Step = 0.2
  End With

  {btnChange} = New Button(Me) As "btnChange"
  With {btnChange}
    .MoveScaled(2,13,32,4)
    .Text = ("Enable-Eigenschaft ändern ') & '...'
  End With

  {Label1} = New Label(Me) As "Label1"
  With {Label1}
    .MoveScaled(36,2,10,4)
    .Text = ("SpinBox")
  End With

  {Label2} = New Label(Me) As "Label2"
  With {Label2}
    .MoveScaled(36,7,10,4)
    .Text = ("SpinBar")
  End With
  End With
End
```

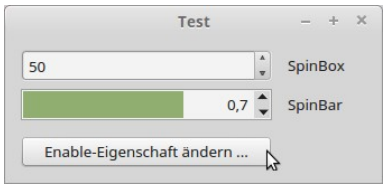

Abbildung 12.2.0.1.1: Anwendungsfenster

Die Form-Klasse ist speziell für die Arbeit in der Gambas-IDE vorgesehen und ihr einziger Zweck besteht darin, auf einfache Weise graphische Anwendungen (GUI) in der IDE schreiben zu können.

12.2.0.2 Formular und Fenster

Wie im ersten Teil bereits formuliert, gibt es eine Form-Klasse und eine Window-Klasse, wobei Form von Window abgeleitet ist und somit Form ein spezielles Window ist. Von einem Formular spricht man im engeren Sinn, wenn es mit dem Formular-Editor in der IDE erzeugt wurde. Die Form-Klasse ist also eine Window-Klasse, die auf Zusammenarbeit mit der Gambas-IDE ausgelegt ist. Ganz technisch ist der Unterschied zwischen Window und Form nur der, dass ein Form-Objekt bereits nach der Erzeugung sein eigener Event-Observer ist. So sind Sie es gewohnt, in der IDE den Quelltext zu einem Formular zu öffnen und zum Beispiel:

```
Public Sub Form_Open()
' ...
End
```

zu implementieren. Das ist aber keine Selbstverständlichkeit. Das geht nur, weil Form-Objekte bei ihrer Erzeugung folgenden Code automatisch ausführen:

```
Object.Attach(Me, Me, "Form")
```

Außerdem sorgt das Form-Objekt dafür – wenn es die Startklasse eines Projektes ist – dass das Anwendungsfenster angezeigt wird. Der Quelltext der Form-Klasse ist in C++ geschrieben, aber formal übersetzt wäre das in Gambas etwa:

```
' Main wird automatisch vom Interpreter in der Startklasse ausgeführt, wenn das Projekt gestartet wird.
Public Sub Main()
  Me.Show()
End
```

Formulare sind aber noch in anderer Hinsicht interessant, nämlich im Hinblick auf den Form-Editor in der IDE und den Gambas-Compiler. Der Form-Editor hinterlegt im Projekt-Ordner .form-Dateien. Wenn der Compiler ein Formular kompiliert, so liest er diese .form-Dateien ein und erstellt aus ihnen *automatisch* Gambas-Code, der in eine versteckte Methode namens '$load" eingetragen wird. Dieser Code erzeugt und positioniert alle Steuerelemente so, wie Sie es im Form-Editor angegeben hatten. Form ist somit eine auf Gambas zugeschnittene Klasse, die nur Komfort in der IDE schaffen soll. Sie können ein Formular zur Entwicklungszeit in der IDE auch als Bild gewordene Form-Klasse auffassen, die Sie zur Laufzeit als Anwendungsfenster auf dem Desktop sehen. Ein zu einem Projekt gehörendes Anwendungsfenster können Sie auch als Frontend zur Verwaltung des Inhalts der graphischen Benutzeroberfläche (graphical user interface, GUI) auffassen.

Unter http://gambaswiki.org/wiki/doc/window finden Sie einige Bemerkungen zum Lebenslauf eines Anwendungsfensters in Gambas.

12.2.0.3 Instanzen einer Form-Klasse erzeugen

Dieser Zweizeiler erzeugt ein neues Form-Objekt, funktioniert wie bei jeder anderen Klasse auch und zeigt es als Anwendungsfenster oder als eingebettetes Fenster in einem Container-Steuer-Element an, wenn der optionale Parameter festgelegt wurde:

```
Dim hForm As Form
hForm = New Form ( [ Parent As Control ] ) As "EventName"
```

Ereignisse werden aber nur dann ausgelöst, wenn auch ein Event-Name vergeben wurde.

Beispiel:

Abbildung 12.2.0.3.1: Zwei eingebettete und 3 Desktop-Fenster

Quell-Text FMain.class:

```
' Gambas class file

Public Sub Form_Open()

  Dim hColorLabel As FColorLabel

  Application.MainWindow = FMain

  FMain.Arrangement = Arrange.Vertical
  FMain.Margin = True
  FMain.Spacing = True
  FMain.Show()

  hColorLabel = New FColorLabel('  EMBEDDED FORM 1", FMain, Color.Yellow)
  hColorLabel = New FColorLabel('  EMBEDDED FORM 2", FMain, Color.White)
' ------------------------------------------------------------------------
  hColorLabel = New FColorLabel('  FREE 1", Null, Color.Green)
  hColorLabel.Show()
  hColorLabel.Move(FMain.X + FMain.W + 14, FMain.Y)
' ------------------------------------------------------------------------
  hColorLabel = New FColorLabel('  FREE 2", Null, Color.Red)
  hColorLabel.Raise()
  hColorLabel.Move(FMain.X + FMain.W + 14, FMain.Y + FMain.H - hColorLabel.H)

End
```

Quell-Text FColorLabel:

```
' Gambas class file

Public Sub _new(Text As String, Optional iColor As Integer)
  lblMe.Text = Text
  lblMe.Border = Border.Solid
  ME.Margin = True
  If Not IsMissing(iColor) Then lblMe.Background = iColor
End
```

12.2.0.4 Zugriff auf Instanzen der gleichen Form-Klasse - FMAIN und ME

Der Einsatz von ME als Verweis auf das aktuelle Form-Objekt ist immer dann notwendig, wenn mindestens zwei Instanzen der gleichen Form-Klasse existieren. Wenn Sie eine Methode aufrufen wollen oder auf eine Eigenschaft oder Variable einer Form-Instanz zugreifen, wie es das folgende Beispiel zeigt:

```
Public Sub _new()
  FSpecial.Center()
End

Public Sub _new()
  ME.Center()
End
```

dann müssen Sie Folgendes beachten: Die Center()-Methode zentriert in beiden Fällen das Anwendungsfenster auf dem Desktop. Es gibt aber einen subtilen Unterschied zwischen FSpecial und ME. Der obere Quelltext benutzt FSpecial, das ist die so genannte "automatische Instanz" des Formulars. ME im Quelltext darunter bezieht sich immer auf das *aktuelle* Form-Objekt! Sie können aus der Klasse FSpecial mehrere Instanzen von FSpecial als Objekt erzeugen. Im oberen Quelltext wird nur das erste Fenster zentriert, wenn eine Instanz von FSpecial erzeugt wird. Bei dem unteren Quelltext dagegen

wird *jede* neue Instanz von FSpecial zentriert. ME gibt eine Referenz auf das aktuelle Form-Objekt zurück, das immer das Objekt repräsentiert, dessen Code gerade vom Interpreter ausgeführt wird. ME ist in der Hinsicht besonders, weil der Gambas-Interpreter den Wert ständig im Hintergrund ändert, weil der Wert in einem anderen Ausführungskontext etwas Anderes repräsentieren kann.

12.2.0.5 Fenster Aktivierung & Deaktivierung

Um das aktive Fenster zu werden, das heißt das Fenster, das durch die Eigenschaft 'Application.ActiveWindow' zurückgegeben wird, muss es entweder ein TopLevel-Fenster sein oder Sie müssen den Form_Activate()-Ereignishandler implementieren, wenn es sich um ein eingebettetes Fenster handelt. Dabei gilt: Wenn ein Steuerelement den Fokus erhält, dann wird dessen Fenster zum aktiven Fenster, sofern dieses Fenster, wie oben erläutert, das aktuelle aktive Fenster werden kann. Wenn das Fenster nicht das aktuelle aktive Fenster werden kann, dann wird versucht, ob es das übergeordnete Fenster dieses Fensters sein kann und so weiter, bis ein Fenster der obersten Ebene erreicht ist. Das Aktivieren eines Fensters sendet ein Deactivate-Ereignis dem alten aktiven Fenster und ein Activate-Ereignis an das Neue. Wird das Anwendungsfenster als Symbol dargestellt, so wird das aktuelle aktive Fenster deaktiviert.

12.2.0.6 Daten-Austausch

Formulare sind 'Create Static'-Klassen. Wenn man den Klassen-Namen wie ein Objekt verwendet, so wird im Hintergrund ein Objekt der Klasse erzeugt und unter dem Klassennamen *öffentlich* verfügbar gemacht. Dieses Objekt nennt man die 'automatische Instanz' der Klasse. Deshalb können Sie von einem Fenster Form2 aus Eigenschaften dieser automatischen Instanz ändern, wenn Form2 eine Referenz auf Form1 besitzt und in Form2.class zum Beispiel der folgende Quell-Text steht:

```
Public Sub Form2Button1_Click()
  Form1.Background = Color.Black
End
```

Auf Steuer-Elemente von Form1 können Sie von Form2 (oder auch umgekehrt) aus nicht zugreifen, da für diese Steuer-Elemente als Standard das Zugriffsrecht 'private' gilt.

Beispiel:

```
If bChanged Then
   Form1.lblTemperaturAnzeige.Text = Format(Form2.fCurTemperature, '##.0 °C")
Endif
```

löst aus diesem Grund einen Fehler aus – auch wenn es auf dem Formular 'FMain' das Steuer-Element *lblTemperaturAnzeige* gibt:

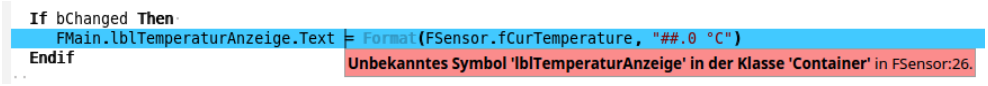

Abbildung 12.2.0.6.1: Fehler-Meldung

Auf die Zugriffsrechte für Steuer-Elemente wird auch in der Gambas-Dokumentation hingewiesen: "*Form controls in Gambas programs are private by default. You can change this by going into the Project Properties dialog and checking the Make form controls public checkbox.*' und gleichzeitig die Möglichkeit genannt, in den Projekt-Eigenschaften eine generelle Änderung des Verhaltens zu erwirken.

Auch im Form-Editor der IDE können Sie für verwendete Steuer-Elemente die Public-Eigenschaft auf True zu setzen. 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.

Im Kapitel 12.2.9 werden Konzepte für den Daten-Transfer zwischen Anwendungsfenstern beschrieben und passende Projekte vorgestellt, welche die Konzepte umsetzen.

12.2.0.7 Startklasse

Jedes Gambas-Projekt braucht eine Startklasse. Diese muss eine gemeinsame (statische) Methode 'Main()' ohne Argumente definieren, die als Startpunkt für die Anwendung fungiert.

Fall 1 – GUI:

Sie können die Startklasse definieren, indem Sie in der IDE im Projekt-Fenster mit der rechten Maustaste auf den Form-Namen klicken und dann im Popup-Menü 'Startklasse' auswählen. Man erkennt das Startformular in der Projekt-Übersicht in der IDE daran, dann es nach rechts herausgerückt ist und vor dem Formularsymbol ein kleines graues Dreieck eingefügt wurde. Da die Startklasse ein Formular ist, muss sie keine Methode *Main* besitzen, da ein Formular bereits eine vorgefertigte Main-Methode mitbringt: Das Formular wird instanziert und angezeigt. In Gambas ist ein Formular sein eigener Event-Observer, so dass Sie seine Events - wie zum Beispiel Resize oder Activate - im Quell-Text der Klasse selbst managen können.

Fall 2 – Gambas-Skript:

Für ein Gambas-Skript, das ohne Anwendungsfenster auskommt, *müssen* Sie die Methode Main() im Quell-Text zwingend implementieren:

```
#!/usr/bin/gbs3

Use "gb.gsl"

Public Sub Main()
 Dim Matrix1, Matrix2 As Matrix
  Matrix1 = [[2, 0], [0, 2]]
  Matrix2 = [[0, 1], [2, 0]]
  Print Matrix1 * Matrix2
End
```

Der Aufruf von *mmult.gbs* – unter diesem Namen wurde der Quelltext gespeichert – mit dem o.a. Inhalt liefert in der System-Konsole das Produkt der beiden Matrizen:

```
hans@linux ~/Test $ gbs3 ./mmult.gbs3
[[0 2][4 0]]
```

12.2.0.8 Methode Form.Load()

Die Methode *Form.Load()* stellt sicher, dass die automatische Instanz der jeweiligen Form-Klasse existiert – sonst würde 'FMain.Load()' das Hauptfenster erstellen. In 99% der Fälle existiert diese automatische Instanz aber schon (in der IDE). Sie wird erzeugt, sobald FMain wie ein Objekt verwendet wird. Insbesondere passiert das genau dann, wenn FMain die Startklasse eines Projekts ist. Sobald das Hauptfenster erzeugt ist, existiert die automatische Instanz und FMain.Load() wäre ohne Effekt.

12.2.0.9 Form.TopLevel

Ein TopLevel-Fenster ist ein Fenster auf dem Desktop. Wenn Sie ein Formular erzeugen, dann ist das ja nur eine Klasse. Sie können daraus – wie man es normalerweise macht – ein Desktop-Fenster erzeugen. Im Gegensatz dazu erzeugen Sie ein Embedded-Fenster als "unechtes" Fenster als Objekt einer Formular-Klasse in einem vorhandenen Container. Sie erhalten dann ein Steuer-Element wie jedes andere auch – das Fenster ist dann nicht mehr top-level.

Quelltext:

```
  Dim hTopLevelWindow, hEmbeddedWindow As Window

' Erzeugt ein Top-Level-Fenster aus der FMain-Klasse
  hTopLevelWindow = New FMain

' Erzeugt ein Nicht-Top-Level-Fenster in einem Container (hier -> hTopLevelWindow)
  hEmbeddedWindow = New FMain(hTopLevelWindow)
```

12.2.0.10 Form.TopOnly

Der folgende Quelltext wird für das existierende Fenster 'FMain' die Eigenschaft 'TopOnly' alternativ festlegen. Im Beispiel wird ein Menü-Eintrag verwendet, um die Eigenschaft 'TopOnly' in zwei Varianten zu setzen. Sie könnten aber auch einen Toggle-Button in der GUI verwenden.

```
PUBLIC SUB mnuOnTop_Click()
  IF mnuOnTop.Value THEN
     FMain.TopOnly = TRUE
```

```
  ELSE
     FMain.TopOnly = FALSE
  ENDIF
END

PUBLIC SUB mnuOnTop_Click()
  FMain.TopOnly = NOT FMain.TopOnly ' Variante 2
END
```

12.2.0.11 Form.Stacking

Stellen Sie sich den Bildschirm mit drei hintereinander liegenden Schichten von Fenstern vor. Fenster der oberen Schicht überdecken immer die der unteren beiden Schichten. Dazwischen befinden sich die normalen Fenster. Die Stacking-Eigenschaft legt fest, zu welcher der drei Schichten das Fenster gehören soll. Die Standard-Schicht ist *Form.Normal*. Beachten Sie, dass die Klasse Form eine spezielle Window-Klasse ist und deshalb die Eigenschaft Window.Stacking erbt. In der Dokumentation zu Form finden Sie deshalb auch:

```
Window.Stacking (gb.qt4)
Property Stacking As Integer
```

Mit dieser Eigenschaft ermitteln Sie die Schicht oder legen die (Desktop-)Schicht des Fensters fest. Die Fenster auf dem Desktop können zu einer von drei Schichten gehören. Die Form.Stacking-Eigenschaft kann einen der folgenden Werte haben oder auf diesen Wert gesetzt werden:

- Window.Above: Das Fenster bleibt über allen anderen Fenstern. Diese Gruppe wird von Desktop-Applets verwendet, die permanent auf dem Desktop sichtbar bleiben sollen.
- Window.Below: Das Fenster bleibt unter allen anderen Fenstern.
- Window.Normal: Das Fenster bleibt mit allen anderen Fenstern auf dem Desktop. Dies ist die Standard-Schicht.

12.2.0.12 Form.Scaled

Die Autoren des Gambas-Buches verwendeten eine Zeit lang zwar die gleiche Gambas-Version, jedoch unter verschiedenen Linux-Systemen (Ubuntu 12.04 LTS und Mint 17). Da die entwickelten Projekte stets sorgsam getestet werden, gab es immer wieder Hinweise zur fehlerhaften Geometrie der Programm-Fenster. Es fiel auf, dass die Programm-Fenster und die Größe aller Steuer-Elemente unter Mint 17 stets kleiner waren als es die Fenster-Bilder in der Projektbeschreibung unter Ubuntu 12.04 LTS darstellten. Der Verkleinerungsfaktor war hinreichend konstant. Die Lösung besteht darin, die Eigenschaft Form.Scaled auf False zu setzen, um zu verhindern, dass die Größe des Formulars und der Steuer-Elemente von der Größe der Standardschrift abhängig ist. Achtung: Diese boolsche Eigenschaft ist virtuell! Sie ist somit nur in der IDE vorhanden und veränderbar und existiert zur Laufzeit nicht mehr. Eine andere Möglichkeit besteht darin die Größe des Formulars und der Steuer-Elemente in der IDE manuell oder zur Laufzeit auf die vorgesehenen Werte anzupassen.

12.2.0.13 Gambas und die Anbindungen an QT4/QT5 und GTK+2/3 als GUI-Toolkit

Sie müssen wissen, dass Gambas sowohl GTK+2/3 als auch QT4/QT5 mit den Komponenten gb.gtk, gb.gtk3 und gb.qt4/5 unterstützt. Es gibt mit gb.gui noch eine weitere Komponente, die je nach dem vorhandenen Desktop eine der ersten drei Komponenten lädt. Dabei kommt es gelegentlich zu Problemen, weil die Schnittstellen der Komponenten zwar kompatibel sind, aber das Verhalten von GTK+ und QT4 sich intern in manchen Teilen unterscheidet. Von der Benutzung von gb.gui sollten Sie deshalb absehen und sich schon beim Anlegen eines neuen Projekts (GUI) für eine der beiden Optionen entscheiden: "QT graphical application" oder "GTK+ graphical application".