Table-Grid-Steuerelement in Dialogen

Wer Anwendungsprogrammierungen in Basic in Verbindung mit Datenbanken bisher erzeugt hat und dafür statt Formularen (also einfachen Dokumenten mit Formularsteuerelemente) eher Dialoge eingesetzt hat, kennt die Problematik, Daten „ordentlich“ darzustellen.

Bisher habe ich hierfür eine Listbox genommen und Datensätze zu einzelnen Strings zusammengebaut. Dabei bestand die Herausforderung darin, alle Spalten zu simulieren, d.h. die Länge der Einträge der einzelnen Datenfelder immer gleich darzustellen.

Gelöst wurde das durch Verwendung eines Fonts mit gleichen Zeichenbreiten (Fixe Fonts) sowie der entsprechenden Aufbereitung der Datensätze. Als Ergebnis waren dann Listen wie in Abb.1 dargestellt möglich – funktional, ihren Zweck erfüllend, aber eigentlich nur eine Notlösung.

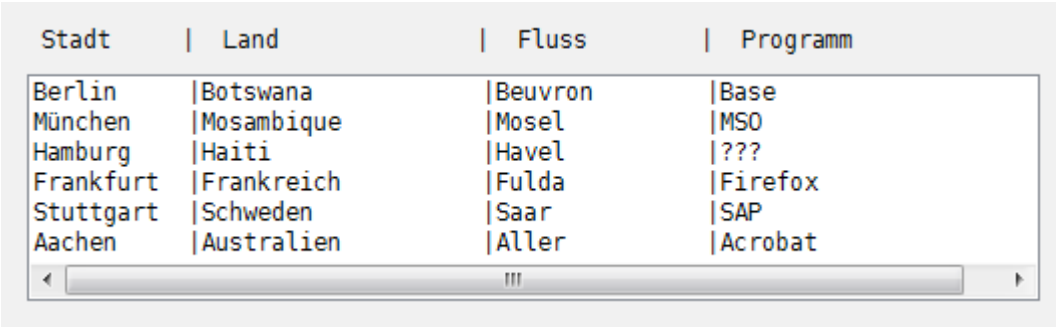

Abbildung 1: Listbox als Tabelle für Datensätze "missbraucht"

Ein Tabellen-Grid Steuerelement wie bei Formularen gibt es im Dialogdesigner nicht, so dass eine solche Lösung bisher entfiel. Mit der Version 3.3 von OpenOffice (und damit auch mit der Version 3.3 von LibreOffice) wurde allerdings das Steuerelement in der API verankert und steht seitdem auf für eigene Programmierungen zur Verfügung.

Leider ist die Nutzung noch ein wenig „holbrig“ und verlangt viel Erfahrung im Umgang mit der API. Im folgenden Artikel beschreibe ich die Möglichkeit, ein solches Table-Grid Steuerelement in einem Dialog zu verwenden und gebe Tipps für den weiteren Einsatz und die Möglichkeiten.

Table-Grid Steuerelement

Zunächst zur Begriffsbestimmung: Ein Tabellen-Steuerelement ist eine Matrix aus Spalten und Zeilen, in deren einzelnen Zellen jeweils Daten untergebracht sind. Das beste Beispiel ist eine Calc-Tabelle. Die Spalten (A, B, C, u.s.w.) sind senkrecht angeordnet, die Zeilen (1,2,3,-..) waagerecht. Jede Zelle kann Inhalte aufnehmen.

Gerade bei Datenbank-Applikationen kommt einem diese Struktur zu Gute: Die Feldnamen werden den Spalten zugeordnet, die Datensätze den Zeilen. Die Tabelle selbst kann dann sehr bequem die Datensätze anzeigen (siehe Abb. 2).

	Stadt	Land	Fluss	Programm
1	Berlin	Botswana	Beuvron	Base
2	München	Mosambique	Mosel	MSO
3	Hamburg	Haiti	Havel	???
4	Frankfurt	Frankreich	Fulda	Firefox
5	Stuttgart	Schweden	Saar	SAP

Abbildung 2: Tabelle - gewünschtes Aussehen

Ein solches Tabellen-Steuerelement besteht somit aus einigen unterschiedlichen Elemente:

Element	**Beschreibung**
Das Kontroll-Element an sich	Quasi das „Rahmenmodell“ mit den wichtigsten Angaben wie Position (X, Y) sowie Breite und Höhe. Dieser Rahmen ist später vom Benutzer nicht veränderbar und integriert sich in den Dialog.
Spaltenköpfe	Die Spaltenköpfe nehmen die Bezeichnung der Spalten auf, sind aber nicht selbst Teil der Inhaltsdaten. Beim vertikalen Scrollen bleiben diese immer am selben Platz.
Spalten	Die Spalten selbst sind einzelne Objekte mit festen Eigenschaften wie Breite, Hintergrundfarbe etc. Der Benutzer kann typischerweise die Spaltenbreite selbst verändern. Ist der Inhalt der Zellen breiter als die Spaltenbreite, wird dieser abgeschnitten (nur in der Anzeige).
Zeilenköpfe	Die Zeilenköpfe nehmen die Bezeichnung der Zeilen auf, sind aber nicht selbst Teil der Inhaltsdaten. Beim horizontalen Scrollen bleiben diese immer am selben Platz.
Zeilen	Die Spalten selbst sind einzelne Objekte, die allerdings nur erscheinen, wenn Datenzeilen vorhanden sind. Die Zeilen können unterschiedliche Eigenschaften (z.B. Hintergrundfarbe) haben.
Vertikale/horizontale Scrollleisten	Eigenständige Objekte, die ein Scrollen mit der Maus ermöglichen, falls aktiviert und falls mehr Inhalte da sind als im angezeigten Kontroll-Element Rahmen angezeigt werden können.

Anders als beim Formular (Tabellensteuerelement) gibt es allerdings keine automatische Navigationsleiste im unteren Bereich.

Alle Details des Services finden sich in der Dokumentation: com.sun.star.awt.grid.UnoControlModel.

Das Steuerelement für den Dialog erzeugen

Nutzt man den Dialog-Designer der Basic-IDE, so ist es direkt nicht möglich, das Kontroll-Element dort zu erzeugen. Will man dennoch auf den Komfort des Designers nicht verzichten, so erzeugt man zunächst den Dialog ohne das Tabelle-Grid Element. Es muss allerdings ein entsprechender Platz freigehalten werden.

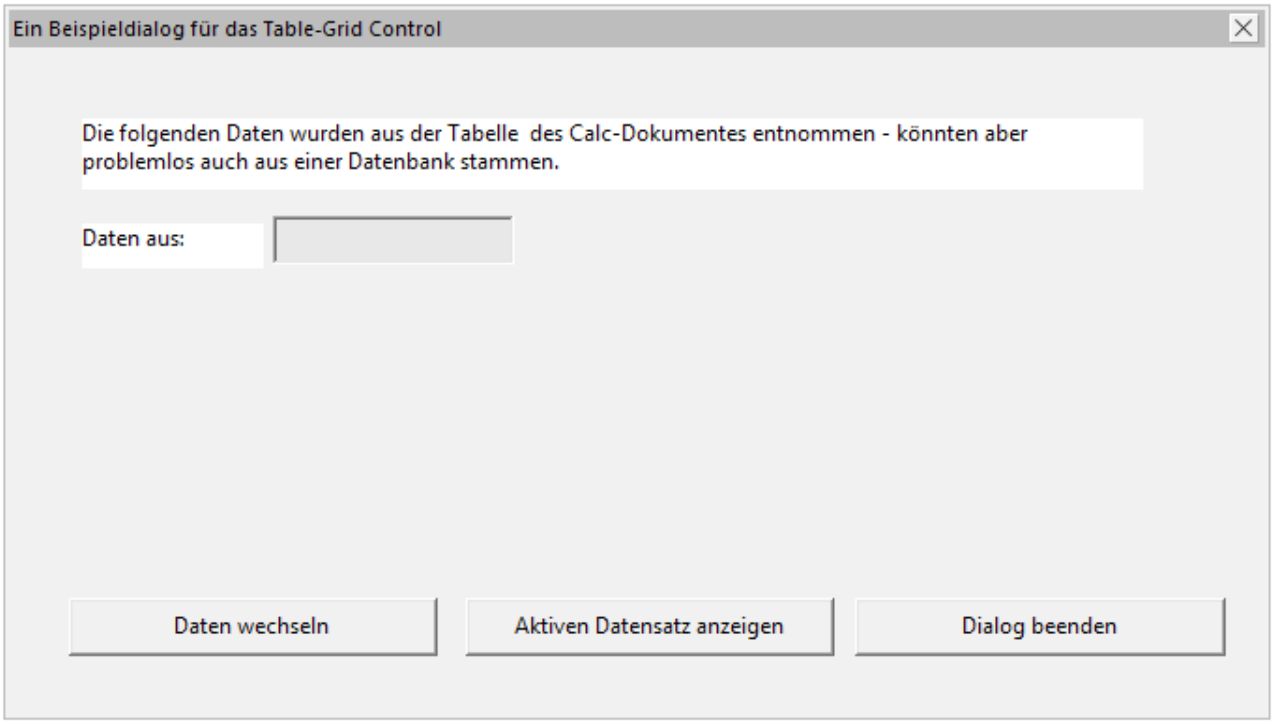

Abbildung 3: Dialog ohne Grid-Element

Um einen optischen Eindruck des Dialoges bereits während der Designphase zu erhalten, macht es Sinn, das Tabellensteuerelement zunächst durch einen Platzhalter zu ersetzen – ich nutze dafür z.B. eine grafische Schaltfläche. Das hat den folgenden Vorteil erstens optisch die Position und Größe gut abschätzen zu können und zweitens liefert mir dann dieses Platzhalter-Element die exakten Koordinaten und die Größe für das spätere Tabellen-Grid Element. Dazu starte ich den Dialog noch mit dem Platzhalter-Element, untersuche dann genau dieses (ich nutze XRAY) und lese die Daten des PosSize-Struct aus. Und diese Daten übernehme ich dann in mein Table-Grid Element.

Kenne ich die Position und Größe des zukünftigen Tabellen-Steuerelementes, so lösche ich das Platzhalter-Element und dann kann die Programmierung losgehen.

Als Grundregel kann gelten: Um ein Tabellen-Steuerelement zu erzeugen, bedarf es dreier einzelnen Services (Objekte), die wiederum alle drei ein Model-Objekt besitzen. Alle müssen erzeugt und zum Schluss dem Dialog hinzugefügt werden.

Dabei dient der com.sun.star.awt.grid Service als „Rahmenmodul“, dem sowohl das Spaltenobjekt als auch das Zeilenobjekt zugewiesen wird. Das Rahmenmodul selbst wird später dem Dialog hinzugefügt.

Doch der Reihe nach:

1 Spaltenobjekt

Als erstes Objekt sollte man das Spaltenobjekt erzeugen. Die Reihenfolge ist zwar faktisch irrelevant, doch entspricht dies der logischen Abfolge.

```
oColumnModel = createUnoService("com.sun.star.awt.grid.DefaultGridColumnModel")
```

Nun gibt es zwar ein (leeres), aber vordefiniertes Spalten-Objekt, das alleine reicht aber noch nicht. Für jede benötigte Spalte muss nun ein eigenes Objekt (ein Spaltenobjekt) erzeugt werden. Diese

wird dann dem Model hinzugefügt. Dabei ist die Reihenfolge des hinzufügens gleich der Reihenfolge der Spalten im Tabellen-Objekt später.

Eine Spalte wird wie folgt erzeugt:

```
oColumn = createUnoService("com.sun.star.awt.grid.GridColumn")
```

Ein solches Spaltenobjekt besitzt nun diverse Eigenschaften, die wichtigsten wären:

Property	Beschreibung
.Title	(string) Der Titel der Spalte – wird im Spaltenkopf angezeigt.
.ColumnWidth	(long) Die breite der Spalte
.MinWidth .MaxWidth	(long) maximale und minimale Spaltenbreite – wirkt nur, wenn die Spaltenbreite vom Benutzer verändert werden kann.
.Resizeable	(boolean) Vorgabe = True Bestimmt, ob die Spaltenbreite vom Benutzer verändert werden kann. Ist der Wert true, kann die Spalte auch automatisch angepasst werden (siehe Flexibility)
.HorizontalAlign	(com.sun.star.style.HorizontalAlignment) Bestimmt die horizontale Ausrichtung innerhalb der Spalte. 0 = linksbündig, 1 = zentriert, 2 = rechtsbündig.
.Flexibility	(long) Bestimmt die Möglichkeit die Spalte automatisch in der Breite zu verändern, wenn das komplette Tabellenelement in seiner Größe geändert wird (automatische Anpassung). Spielt bei Basic-Dialogen eine untergeordnete Rolle, da es keine Möglichkeit des Benutzerdefinierten Anpassens des Kontrollelementes an sich gibt. Vorgabe ist 1 (true), was bedeutet, dass alle Spalten an den vorhandenen Platz automatisch gleichmäßig angepasst werden. Empfehlung: Immer auf 0 stellen (false), wenn eigene Breitenvorgaben erfolgen (ColumnWidth).
.HelpText	(string) ein Hilfetext, der typischerweise als Tooltipp angezeigt wird, wenn die Maus über dem Spaltenkopf steht.

Sind die Eigenschaften entsprechend eingestellt, so kann die Spalte nun dem Spaltenobjekt zugefügt werden:

```
oColumnModel.addColumn(oColumn)
```

Dieser Vorgang muss jetzt für jede einzelne Spalte erfolgen.

Allerdings kann man das Ganze auch als Schleife realisieren (siehe Beispiel)

2 Datenobjekt (Datenzeilen)

Neben dem Spaltenobjekt benötigt man nun noch ein Zeilenobjekt – also die Daten an sich. Fehlt das Datenobjekt (die Zeilen), dann bleibt das Controllelement eben leer – mit dem Spaltenobjekt stehen wenigstens die Spaltenüberschriften da.

Auch das Zeilenobjekt kennt Zeilenköpfe, also die Bezeichnung der jeweiligen Zeile. An sich muss hier nichts eingetragen werden – oder man nummeriert die Zeilen einfach durch (wie im Beispiel).

Manchmal werden auch Grafiken eingeblendet (z.B. einen Pfeil für die aktive Datenzeile) – aber das ist eigentlich erst der zweite Schritt.

Erzeugen wir zunächst ein entsprechendes Basis Datenmodel:

```
oDataModel = createUnoService("com.sun.star.awt.grid.DefaultGridDataModel")
```

Diesem Objekt nun können einzelne Zeilen (Datenzeilen) hinzugefügt werden:

```
oDataModel.addRow("Zeilenkopf“, aDatenarray())
```

Der Zeilenkopf ist die Zeilenkopf-Beschriftung, der Datenarray eine Liste der Daten. Dabei muss die Dimension der Liste natürlich zu dem späteren Spaltenmodell passen – also entsprechend vorsehen.

Die Zeile wird am Ende der schon vorhandenen Zeilen eingefügt!

Natürlich ist es mühsam, diesen Vorgang für jede einzelne Zeile durchzuführen. Die Methode

```
oDataModel.addRows( aZeilenköpfe(), aDaten())
```

ermöglicht es, auch mehrere Zeilen gleichzeitig hinzuzufügen. Dabei müssen die Zeilenkopf-Bezeichnungen in einer Liste (array) übergeben werden (Dimension entsprechend der Zeilen-Anzahl!) und die Daten als Array von Arrays, also einer Liste von Datensätzen, jeweils als Liste.

Auch hier müssen die Dimensionen passen!

3 TabellenGrid-Controlelement

Zum Schluss nun wird das Grid-Controlelement an sich erzeugt:

Wichtig dabei ist, dass dieses bereits als Instanz des Dialoges (hier: oDlg) erzeugt wird – der Dialog muss also vorher erzeugt worden sein!

```
oGridModel = oDlg.Model.createInstance("com.sun.star.awt.grid.UnoControlGridModel")
REM globale Eigenschaften zuweisen
With oGridModel
   .Name = "myGrid"
  .ShowColumnHeader = True
  .ShowRowHeader = True
  .HScroll = false
  .VScroll = True
  .Sizeable = True
  .Step = 0
  .GridDataModel = oDataModel              'Das Daten-Model wird zugewiesen
  .ColumnModel = oColumnModel              'Das Spalten-Model wird zugewiesen
End with
```

Die meisten Eigenschaften sprechen für sich. Hier aufgeführt sind nur die wichtigsten – zahlreiche weitere findet man in der Dokumentation vom Service "com.sun.star.awt.grid.UnoControlGridModel"

So lässt sich beispielsweise auch einstellen, ob Trennlinien gezeigt werden (UseGridLines = true), welche Farbe sie haben und so weiter.

Auch die Text- und Hintergrundfarben der Spalten- und Zeilenköpfe lässt sich anpassen ebenso wie der Font und vieles mehr.

Hat man das Grid-Modell erzeugt und diesem das Datenmodel und das Spaltenmodel zugewiesen, kann man es nun dem Dialog an sich zuweisen:

```
oGridControl = createUnoService("com.sun.star.awt.grid.UnoControlGrid")

oGridControl.setModel(oGridModel)    'das Model zuweisen

oDlg.addControl("tab_g1", oGridControl)    'dem Dialog hinzufügen
```

Würde man den Dialog jetzt starten, wäre immer noch nichts zusehen. Das Kontrollelement ist zwar Teil des Dialoges – hat aber noch keine Größe und Position. Diese Eigenschaften können aber erst gesetzt werden, wenn das Kontrollelement Teil des Dialoges ist. Also jetzt. Die Werte haben Sie idealerweise von Ihrem ursprünglich verwendeten Platzhalter übernommen – ansonsten hilft nur probieren.

```
oGridControl.setPosSize(39,143,591,122, com.sun.star.awt.PosSize.POSSIZE)
```

SetPosSize erwartet fünf Parameter: X-Koordinate, Y-Koordinate, Breite, Höhe sowie eine Konstante, die bestimmt, welche der vier Werte geändert werden soll (in dem Fall übernommen). Die Konstante POSSIZE ändert sowohl die Größe als auch die Position.

Nun kann der Dialog angezeigt werden – Das Tabellen Kontrollelement ist vorhanden und die Daten werden dargestellt. Einzelne Zeilen können selektiert werden.

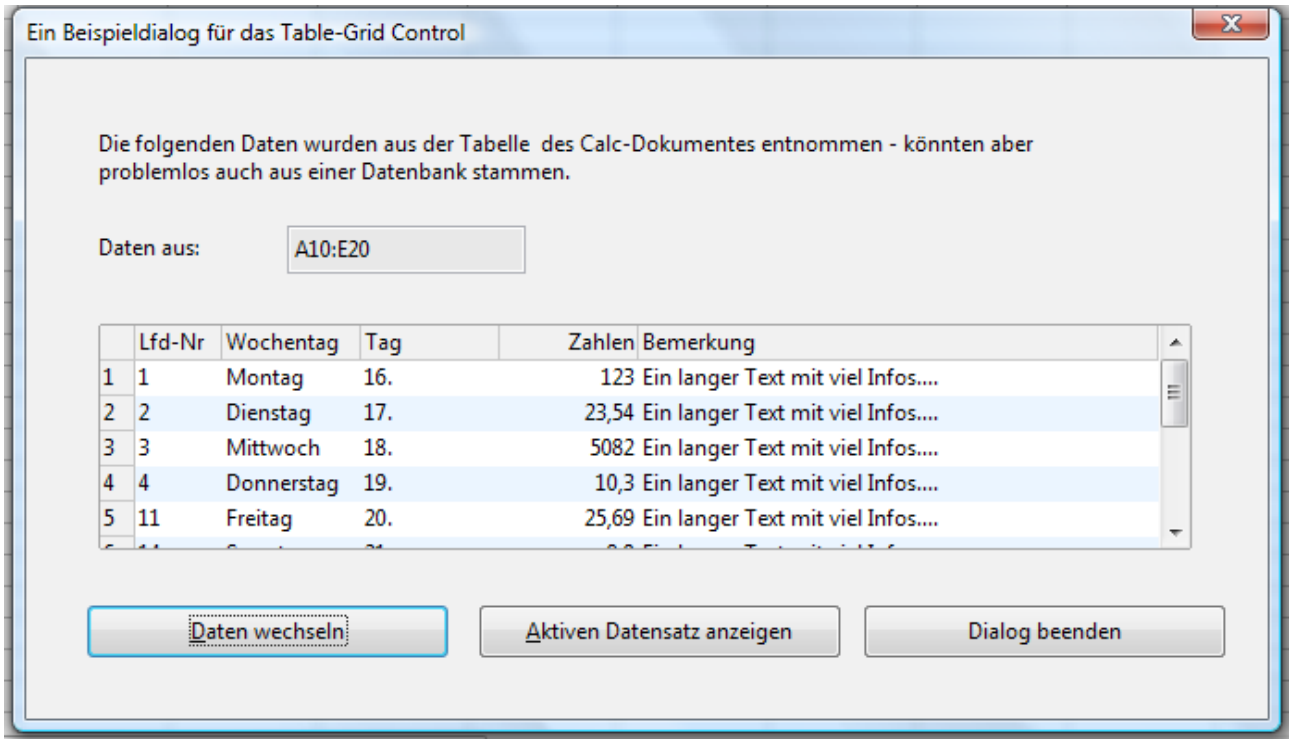

Abbildung 4: Der fertige Dialog

Auslesen der Daten

In der Basisversion lässt sich nur eine Datenreihe markieren. In der Regel ist dies auch ausreichend, benötigt man andere Möglichkeiten, so kann dies über die Eigenschaften entsprechend angepasst werden.

Wird eine Zeile (Datenreihe) markiert (also selektiert), so kann man entweder direkt mit einem Listener auf das Event reagieren (SelectionChangeListener) oder aber manuell die entsprechenden Daten auslesen. Der Listener reagiert sofort – was sich in der Praxis oft als „unschön“ erweist, insofern ist die manuelle Variante die eher anzutreffende.

```
if oDlg.getControl("tab_g1").hasSelectedRows() then _

 aZeile =
oDlg.getControl("tab_g1").model.GridDataModel.getRowData(oDlg.getControl("tab_g1").getCurrentRow)
```

Dieser Code liefert den markierten Datensatz als Datenarray – die Auswertung und Aufbereitung kann beginnen.

Es gibt zwar auch Methoden, nur eine spezielle Zelle auszulesen:

```
getCellData(Spaltenindex, Zeilenindex)
```

im Praxistest erweisen sich jedoch die Indices als „tückisch“ - während die erste Selektion meist klappt, sind alle folgenden irgendwie „verschoben“, passen jedenfalls nicht mehr zum erwarteten Ergebnis.

Tipps und Tricks

Es ist möglich, die jeweiligen Modelle des Table-Grid Controls zur Laufzeit auszutauschen. Typischerweise passiert dies, wenn neue Daten eingelesen und dargestellt werden müssen. Dazu werden dann im Datenmodel (erreichbar über oDlg.getControl("tab_g1").model.GridDataModel) mit den dort vorhandenen Methoden Datenreihen hinzugefügt oder entfernt.

Methode	Beschreibung
addRow(Zeilenkopf, aDaten)	Fügt eine zusätzliche Datenreihe (Zeile) am Ende an .
addRows(aZeilenköpfe(), aDaten)	Fügt eine Anzahl von Datensätzen (Array von Arrays) der vorhandenen Liste am Ende hinzu.
InsertRow(iIndex, Zeilenkopf, aDaten)	Fügt eine zusätzliche Datenreihe an der Position iIndex (Zeilenindex, Start bei 0) der Liste ein.
InsertRows(iIndex, aZeilenköpfe, aDaten)	Fügt eine Anzahl von Datensätzen (Array von Arrays) an der Position iIndex (Zeilenindex, Start bei 0) der Liste ein.
removeRow(iZeilenindex)	Entfernt eine Datenzeile – definiert durch den Zeilenindex.
RemoveAllRows()	Entfernt alle Datenzeilen.

Möchte man also die Daten austauschen, entfernt man zunächst alle vorhandenen Datenzeilen (removeAllRows()) und fügt dann die neuen Datensätze wie oben beschrieben ein.

Dabei wird das Spaltenmodell nicht verändert – dieser Ablauf ist stabil und effizient.

Man kann allerdings auch sowohl das Spaltenmodell als auch das Datenmodell austauschen und somit das TableGrid-Controll Element universeller verwenden. Das beigelegte Beispiel zeigt diesen

Vorgang.

Es ist jedoch nicht wirklich zu empfehlen. Da die Fenstermanager das Neuzeichnen des Dialoges nicht automatisch durchführen wenn das Modell geändert wurde sondern nur den Teil rendert, der tatsächlich geändert wurde, verbleiben oft unschöne optische Reste des vorherigen Spaltenmodells noch sichtbar. Sie sind zwar funktionslos, stören aber optisch (insbesondere, wenn im ausgetauschten neuen Spaltenmodell weniger und kleinere Spalten vorhanden sind). Leider konnte ich noch keine Funktion finden, die ein Neuzeichnen des kompletten Controllelementes bewirken.

Es empfiehlt sich somit, pro verwendeten Spaltenmodells jeweils ein Tabellenkontroll-Element vorzusehen. Sollen diese auf der gleichen Position liegen, so verwenden Sie die Step-Eigenschaft des Dialoges – und platzieren die Controlls auf unterschiedlichen Seiten.

Fazit

Wenn auch noch etwas aufwendig in der Konstruktion stellt das Tabellen-Kontrollelement für Dialoge doch eine hervorragende Basis dar, Datenbank-Applikationen per Basic zu programmieren.

Die beifügte Beispieldatei Bsp_TableGridDlg.ods dient zum Ausprobieren und zum Code Studieren. Der passende Makro-Code befindet sich in der Bibliothek Bsp_TableGrid. Hier wurden bereits die einzelnen Schritte in eigene Funktionen gekapselt um so die Wiederverwertbarkeit zu erhöhen.

* * * * *