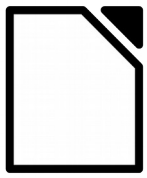


LibreOffice
The Document Foundation

Base

Kapitel 9
Makros

Copyright

Mitwirkende/Autoren

Robert Großkopf
Jost Lange
Jochen Schiffers
Jürgen Thomas
Michael Niedermair

Rückmeldung (Feedback)

Kommentare oder Vorschläge zu diesem Dokument können Sie in deutscher Sprache an die Adresse *discuss@de.libreoffice.org* senden.

Vorsicht 	Alles, was an eine Mailingliste geschickt wird, inklusive der E-Mail-Adresse und anderer persönlicher Daten, die die E-Mail enthält, wird öffentlich archiviert und kann nicht gelöscht werden. Also, schreiben Sie mit Bedacht!

Datum der Veröffentlichung und Softwareversion

Veröffentlicht am 30.03.2019. Basierend auf der LibreOffice Version 6.2.

Anmerkung für Macintosh Nutzer

Einige Tastenbelegungen (Tastenkürzel) und Menüeinträge unterscheiden sich zwischen der Macintosh Version und denen für Windows- und Linux-Rechnern. Die unten stehende Tabelle gibt Ihnen einige grundlegende Hinweise dazu. Eine ausführlichere Aufstellung dazu finden Sie in der Hilfedatei des jeweiligen Moduls.

Windows/Linux	***entspricht am Mac***	***Effekt***
Menü-Auswahl **Extras → Optionen**	**LibreOffice → Einstellungen**	Zugriff auf die Programmoptionen
Rechts-Klick	Control+Klick	Öffnen eines Kontextmenüs
Ctrl (Control) oder *Strg* (Steuerung)	⌘ *(Command)*	Tastenkürzel in Verbindung mit anderen Tasten
F5	*Shift*+⌘+*F5*	Öffnen des Dokumentnavigator-Dialogs
F11	⌘+*T*	Öffnen des Formatvorlagen-Dialogs

Inhalt

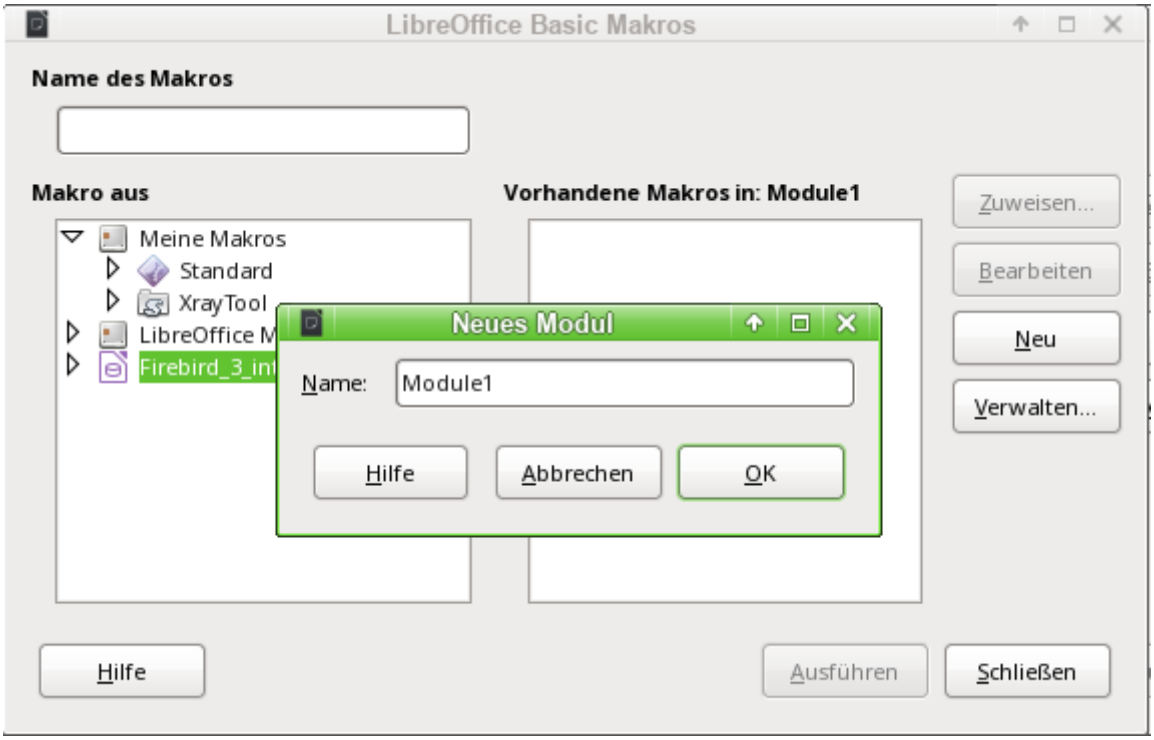
Allgemeines zu Makros

Prinzipiell kommt eine Datenbank unter Base ohne Makros aus. Irgendwann kann aber das Bedürfnis kommen,

- bestimmte Handlungsschritte zu vereinfachen (Wechsel von einem Formular zum anderen, Aktualisierung von Daten nach Eingabe in einem Formular …),
- Fehleingaben besser abzusichern,
- häufigere Aufgaben zu automatisieren oder auch
- bestimmte SQL-Anweisungen einfacher aufzurufen als mit dem separaten SQL-Editor.

Es ist natürlich jedem selbst überlassen, wie intensiv er/sie Makros in Base nutzen will. Makros können zwar die Bedienbarkeit verbessern, sind aber auch immer mit geringen, bei ungünstiger Programmierung auch stärkeren Geschwindigkeitseinbußen des Programms verbunden. Es ist immer besser, zuerst einmal die Möglichkeiten der Datenbank und die vorgesehenen Einstellmöglichkeiten in Formularen auszureizen, bevor mit Makros zusätzliche Funktionen bereitgestellt werden. Makros sollten deshalb auch immer wieder mit größeren Datenbanken getestet werden, um ihren Einfluss auf die Verarbeitungsgeschwindigkeit abschätzen zu können.

Makros werden über den Weg **Extras → Makros → Makros verwalten → LibreOffice Basic...** erstellt. Es erscheint ein Fenster, das den Zugriff auf alle Makros ermöglicht. Makros für Base werden meistens in dem Bereich gespeichert, der dem Dateinamen der Base-Datei entspricht.



Über den Button *Neu* im Fenster «LibreOffice Basic Makros» wird ein zweites Fenster geöffnet. Hier wird lediglich nach der Bezeichnung für das Modul (Ordner, in dem das Makro abgelegt wird) gefragt. Der Name kann gegebenenfalls auch noch später geändert werden.

Sobald dies bestätigt wird, erscheint der Makro-Editor und auf seiner Eingabefläche wird bereits der Start und das Ende für eine Prozedur angegeben:

```
REM  *****  BASIC  *****

Sub Main

End Sub
```

Um Makros, die dort eingegeben wurden, nutzen zu können, sind folgende Schritte notwendig:

- Unter **Extras → Optionen → Sicherheit → Makrosicherheit** ist die Sicherheitsstufe auf «Mittel» herunter zu stellen. Gegebenenfalls kann auch zusätzlich unter «Vertrauenswürdige Quellen» der Pfad angegeben werden, in dem eigene Dateien mit Makros liegen, um spätere Nachfragen nach der Aktivierung von Makros zu vermeiden.
- Die Datenbankdatei muss nach der Erstellung des ersten Makro-Moduls einmal geschlossen und anschließend wieder geöffnet werden.

Einige Grundprinzipien zur Nutzung des Basic-Codes in LibreOffice:

- Zeilen haben keine Zeilenendzeichen. Zeilen enden mit einem festen Zeilenumbruch.
- Zwischen Groß- und Kleinschreibung wird bei Funktionen, reservierten Ausdrücken usw. nicht unterschieden. So ist z.B. die Bezeichnung «String» gleichbedeutend mit «STRING» oder auch «string» oder eben allen anderen entsprechenden Schreibweisen. Groß- und Kleinschreibung dienen nur der besseren Lesbarkeit.
- Eigentlich wird zwischen Prozeduren (beginnend mit **SUB**) und Funktionen (beginnend mit **FUNCTION**) unterschieden. Prozeduren sind ursprünglich Programmabschnitte ohne Rückgabewert, Funktionen können Werte zurückgeben, die anschließend weiter ausgewertet werden können. Inzwischen ist diese Unterscheidung weitgehend irrelevant; man spricht allgemein von Methoden oder Routinen – mit oder ohne Rückgabewert. Auch eine Prozedur kann einen Rückgabewert (außer «Variant») erhalten; der wird einfach in der Definition zusätzlich festgelegt:

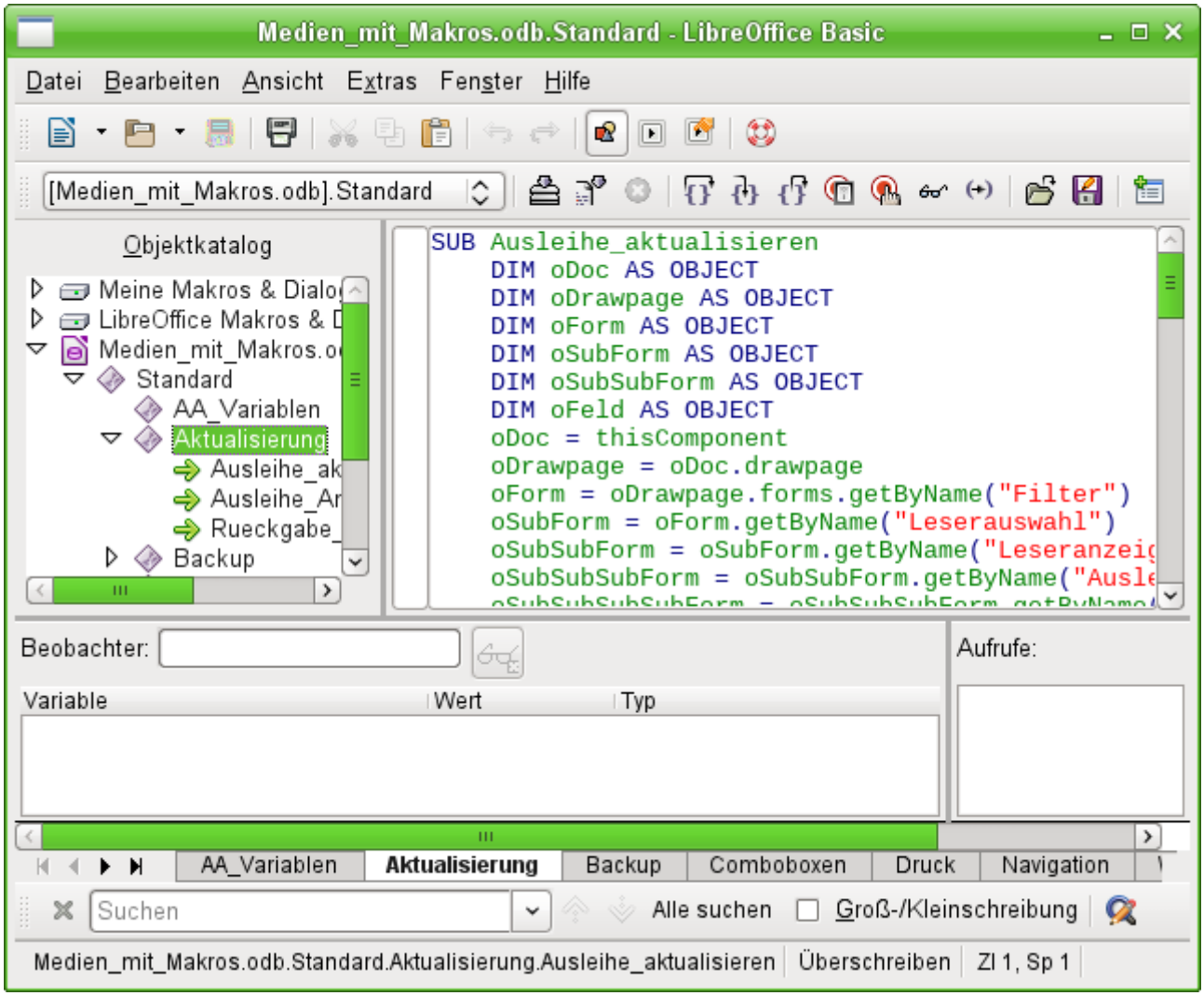
```
SUB myProcedure AS INTEGER
END SUB
```

Zu weiteren Details siehe auch das Handbuch 'Erste Schritte Makros mit LibreOffice'.

Hinweis	Makros in diesem Kapitel sind entsprechend den Vorgaben aus dem Makro-Editor von LibreOffice eingefärbt:
	`Makro-Bezeichner`
	`Makro-Kommentar`
	`Makro-Operator`
	`Makro-Reservierter-Ausdruck`
	`Makro-Zahl`
	`Makro-Zeichenkette`

Hinweis	Bezeichner können frei gewählt werden, sofern sie nicht einem reservierten Ausdruck entsprechen. Viele Makros sind in dieser Anleitung mit an die deutsche Sprache angelehnten Bezeichnern versehen. Dies führte bei der englischsprachigen Übersetzung allerdings zu zusätzlichen Problemen. Deshalb sind die Bezeichner in neueren Makros an die englische Sprache angelehnt.

Der Makro-Editor



Der Objektkatalog auf der linken Seite zeigt alle zur Zeit verfügbaren Bibliotheken und darin Module an, die über ein Ereignis aufgerufen werden können. «Meine Makros & Dialoge» ist für alle Dokumente eines Benutzers verfügbar. «LibreOffice Makros & Dialoge» sind für alle Benutzer des Rechners und auch anderer Rechner nutzbar, da sie standardmäßig mit LibreOffice installiert werden. Hinzu kommen noch die Bibliotheken, die in dem jeweiligen Dokument, hier «Medien_mit_Makros.odb», abgespeichert sind.

Prinzipiell ist es zwar möglich, aus allen verfügbaren Bibliotheken die Module und die darin liegenden Makros zu nutzen. Für eine sinnvolle Nutzung empfiehlt es sich aber nicht, Makros aus anderen Dokumenten zu nutzen, da diese eben nur bei Öffnung des entsprechenden Dokumentes verfügbar sind. Ebenso ist es nicht empfehlenswert, Bibliotheken aus «Meine Makros & Dialoge» einzubinden, wenn die Datenbankdatei auch an andere Nutzer weitergegeben werden soll. Ausnahmen können hier Erweiterungen («Extensions») sein, die dann mit der Datenbankdatei weiter gegeben werden.

In dem Eingabebereich wird aus dem Modul «Aktualisierung» die Prozedur «Ausleihe_aktualisieren» angezeigt. Eingegebene Zeilen enden mit einem Return. Groß- und Kleinschreibung sowie Einrückung des Codes sind in Basic beliebig. Lediglich der Verweis auf Zeichenketten, z.B. "Filter", muss genau der Schreibweise in dem Formular entsprechen.

Makros können schrittweise für Testzwecke durchlaufen werden. Entsprechende Veränderungen der Variablen werden im Beobachter angezeigt.

Benennung von Modulen, Dialogen und Bibliotheken

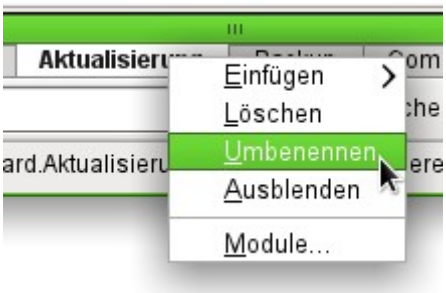
Die Benennung von Modulen, Dialogen und Bibliotheken sollte erfolgen, bevor irgendein Makro in die Datenbank eingebunden wird. Sie definieren schließlich den Pfad, in dem das auslösende Ereignis nach dem Makro sucht.

Innerhalb einer Bibliothek kann auf alle Makros der verschiedenen Module zugegriffen werden. Sollen Makros anderer Bibliotheken genutzt werden, so müssen diese extra geladen werden:

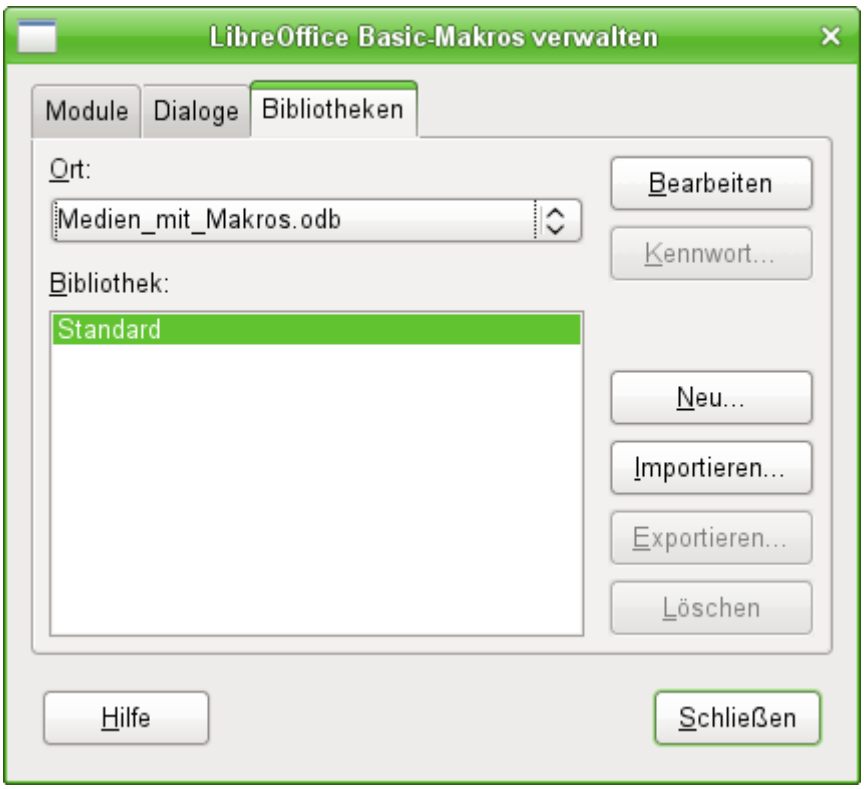
```
GlobalScope.BasicLibraries.LoadLibrary("Tools")
```

lädt die Bibliothek «Tools», die eine Bibliothek von LibreOffice Makros ist.

Über **Extras → Makros verwalten → LibreOffice Basic → Verwalten** kann der obige Dialog aufgerufen werden. Hier können neue Module und Dialoge erstellt und mit einem Namen versehen werden. Die Namen können allerdings nicht hier, sondern nur in dem Makroeditor selbst verändert werden.



Im Makroeditor wird mit einem rechten Mausklick auf die Reiter mit der Modulbezeichnung direkt oberhalb der Suchleiste ein Kontextmenü geöffnet, das u.a. die Änderung des Modulnamens ermöglicht.



Neue Bibliotheken können innerhalb der Base-Datei angelegt werden. Die Bezeichnung «Standard» der ersten erstellten Bibliothek lässt sich nicht ändern. Die Namen der weiteren Bibliotheken sind frei wählbar, anschließend aber auch nicht änderbar. In eine Bibliothek können Makros aus anderen Bibliotheken importiert werden. Sollte also der dringende Wunsch bestehen, eine andere Bibliotheksbezeichnung zu erreichen, so müsste eine neue Bibliothek mit diesem Namen erstellt werden und sämtlicher Inhalt der alten Bibliothek in die neue Bibliothek exportiert werden. Dann kann anschließend die alte Bibliothek gelöscht werden.

Tipp	Bei der Bibliothek «Standard» ist es nicht möglich, ein Kennwort zu setzen. Sollen Makros vor den Blicken des normalen Nutzers verborgen bleiben, so muss dafür eine neue Bibliothek erstellt werden. Diese lässt sich dann mit einem Kennwort schützen.

Makros in Base

Makros benutzen

Der «direkte Weg» über **Extras → Makros → Makros ausführen** ist zwar auch möglich, aber bei Base-Makros nicht üblich. Ein Makro wird in der Regel einem Ereignis zugeordnet und durch dieses Ereignis gestartet.

- Ereignisse eines Formular
- Bearbeitung einer Datenquelle innerhalb des Formulars
- Wechsel zwischen verschiedenen Kontrollfeldern
- Reaktionen auf Maßnahmen innerhalb eines Kontrollfelds

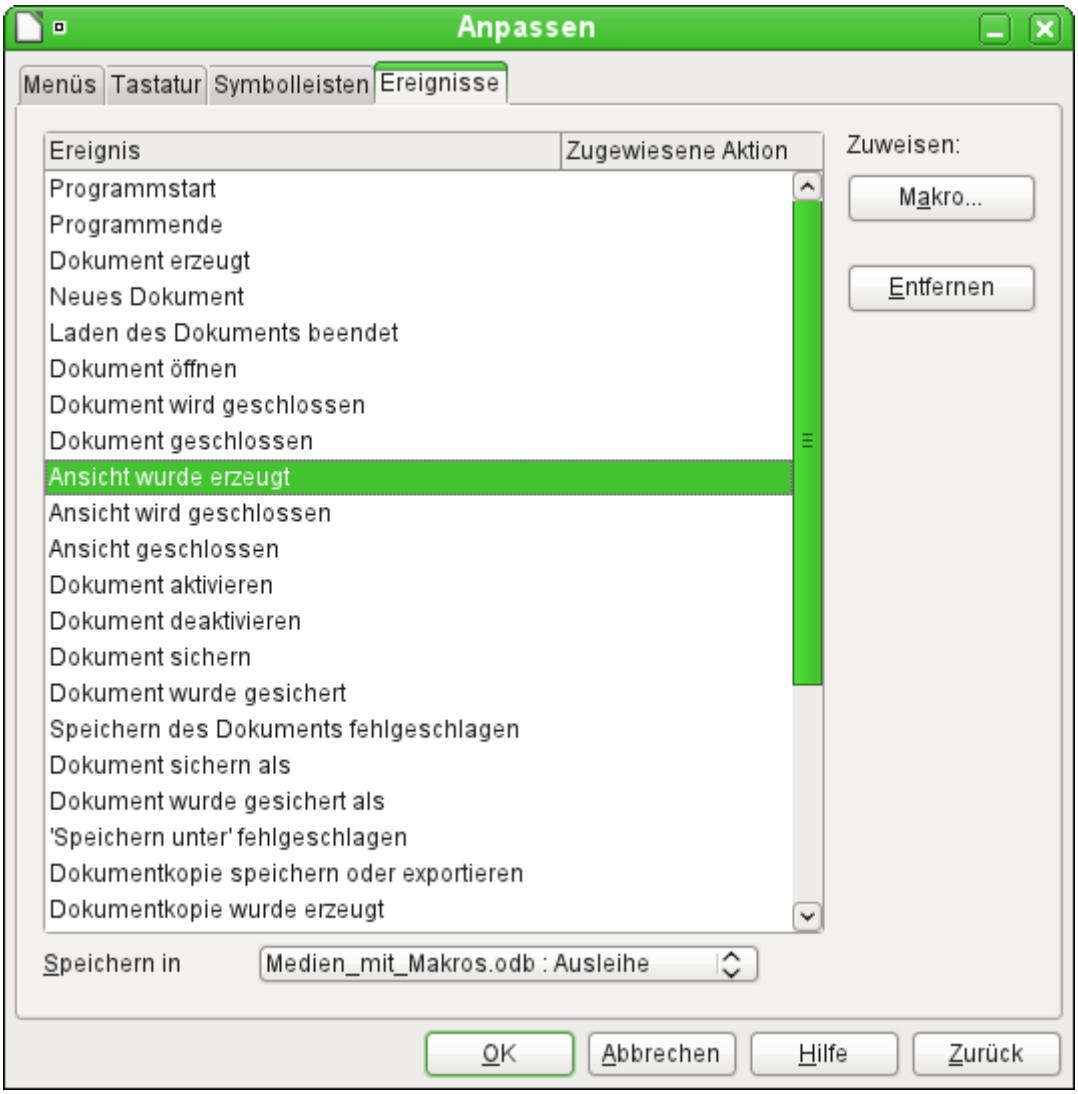
Der «direkte Weg» ist vor allem dann nicht möglich – auch nicht zu Testzwecken –, wenn eines der Objekte **`thisComponent`** (siehe den Abschnitt «*Zugriff auf das Formular*») oder **`oEvent`** (siehe den Abschnitt «*Zugriff auf Elemente eines Formulars*») benutzt wird.

Makros zuweisen

Damit ein Makro durch ein Ereignis gestartet werden kann, muss es zunächst definiert werden (siehe den einleitenden Abschnitt «*Allgemeines zu Makros*»). Dann kann es einem Ereignis zugewiesen werden. Dafür gibt es vor allem zwei Stellen.

Ereignisse eines Formulars beim Öffnen oder Schließen des Fensters

Maßnahmen, die beim Öffnen oder Schließen eines Formulars erledigt werden sollen, werden so registriert:



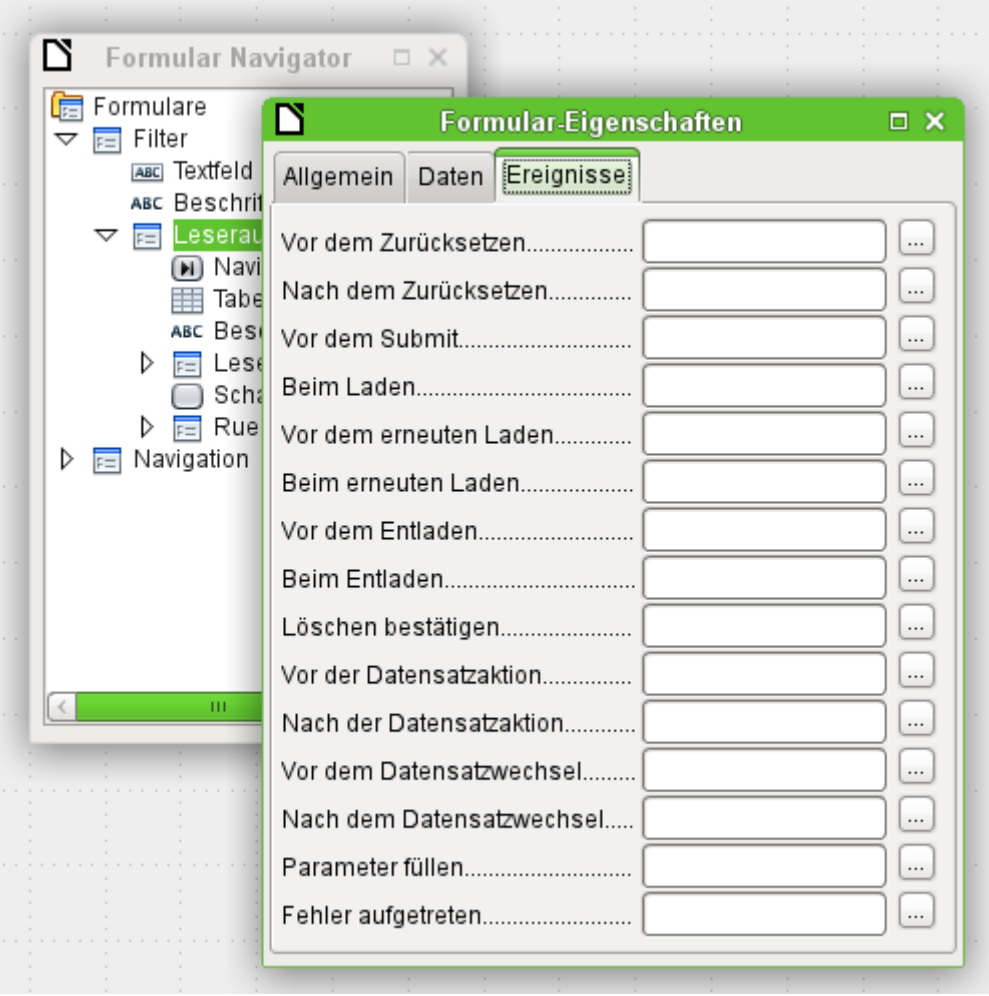
- Rufen Sie im Formularentwurf über **Extras → Anpassen** das Register *Ereignisse* auf.
- Wählen Sie das passende Ereignis aus. Bestimmte Makros lassen sich nur starten, wenn **Ansicht wurde erzeugt** gewählt ist. Andere Makros wie z.B. das Erzeugen eines Vollbild-Formulars kann über **Dokument öffnen** gestartet werden.
- Suchen Sie über die Schaltfläche *Makro* das dafür definierte Makro und bestätigen Sie diese Auswahl.
- Unter *Speichern in* ist das Formular anzugeben.

Dann kann diese Zuweisung mit *OK* bestätigt werden.

Die Einbindung von Makros in das Datenbankdokument erfolgt auf dem gleichen Weg. Nur stehen hier teilweise andere Ereignisse zur Wahl.

Ereignisse eines Formulars bei geöffnetem Fenster

Nachdem das Fenster für die gesamten Inhalte des Formulars geöffnet wurde, kann auf die einzelnen Elemente des Formulars zugegriffen werden. Hierzu gehören auch die dem Formular zugeordneten Formularelemente.



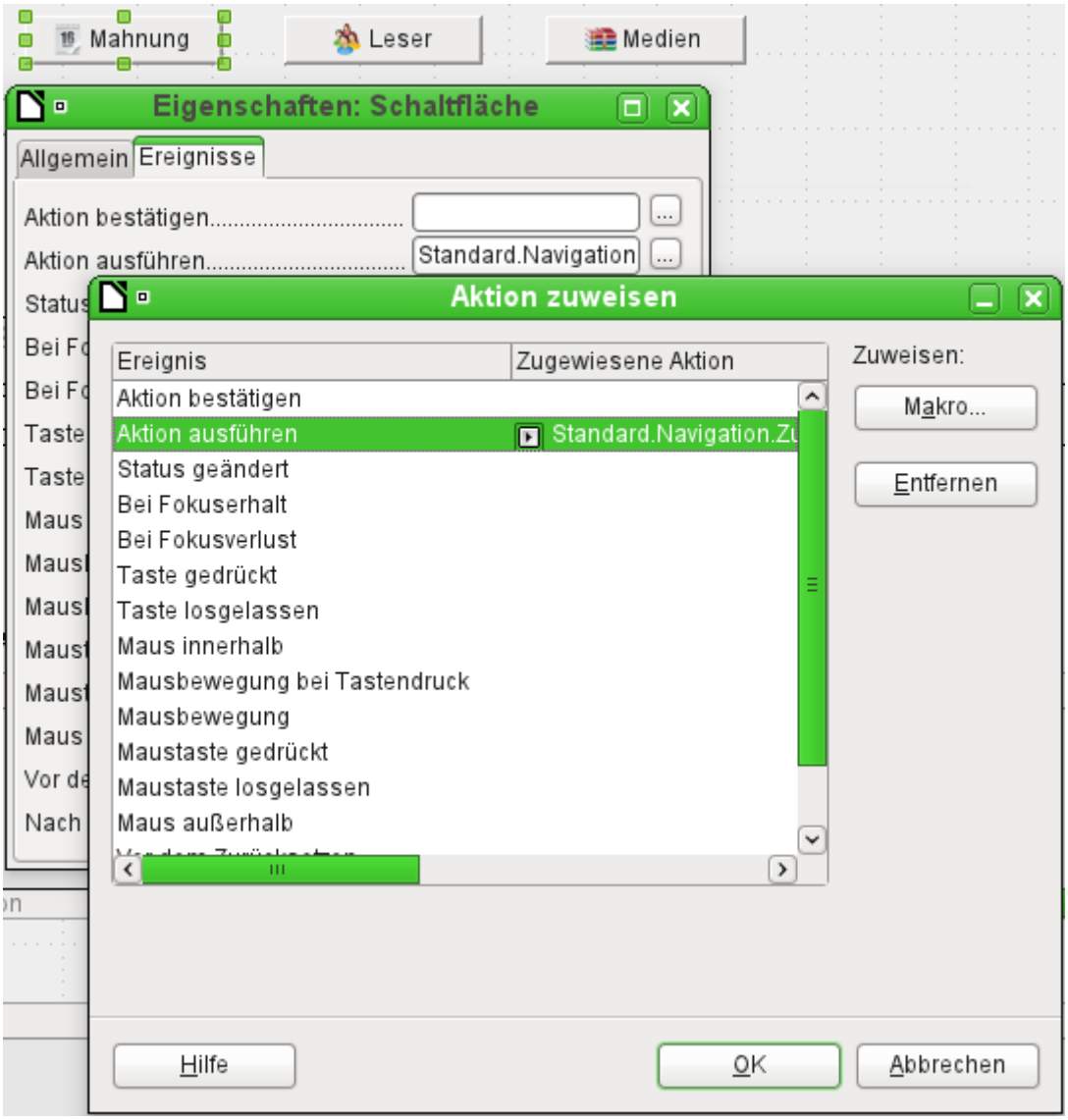
Die Formularelemente können, wie in obigem Bild, über den Formularnavigator angesteuert werden. Sie sind genauso gut über jedes einzelne Kontrollfeld der Formularoberfläche über das Kontextmenü des Kontrollfeldes zu erreichen.

Die unter **Formular-Eigenschaften → Ereignisse** aufgezeigten Ereignisse finden statt während das Formularfenster geöffnet ist. Sie können für jedes Formular oder Unterformular des Formularfensters separat ausgewählt werden.

Hinweis	Der Gebrauch des Begriffes «Formular» ist bei Base leider nicht eindeutig. Der Begriff wird zum einen für das Fenster benutzt, das zur Eingabe von Daten geöffnet wird. Zum anderen wird der Begriff für das Element genutzt, das mit diesem Fenster eine bestimmte Datenquelle (Tabelle oder Abfrage) verbindet. Es können auf einem Formularfenster sehr wohl mehrere Formulare mit unterschiedlichen Datenquelle untergebracht sein. Im Formularnavigator steht zuerst immer der Begriff «Formulare», dem dann in einem einfachen Formular lediglich ein Formular untergeordnet wird.

Ereignisse innerhalb eines Formulars

Alle anderen Makros werden bei den Eigenschaften von Teilformularen und Kontrollfeldern über das Register *Ereignisse* registriert.



- Öffnen Sie (sofern noch nicht geschehen) das Fenster mit den Eigenschaften des Kontrollfelds.
- Wählen Sie im Register *Ereignisse* das passende Ereignis aus.
  - Um die Datenquelle zu bearbeiten, gibt es vor allem die Ereignisse, die sich auf *Datensatz* oder *Aktualisieren* oder *Zurücksetzen* beziehen.
  - Zu Schaltflächen oder einer Auswahl bei Listen- oder Optionsfeldern gehört in erster Linie das Ereignis *Aktion ausführen*.
  - Alle anderen Ereignisse hängen vom Kontrollfeld und der gewünschten Maßnahme ab.
- Durch einen Klick auf den rechts stehenden Button … wird das Fenster «Aktion zuweisen» geöffnet.
- Über die Schaltfläche *Makro* wird das dafür definierte Makro ausgewählt.

Über mehrfaches *OK* wird diese Zuweisung bestätigt.

Bestandteile von Makros

In diesem Abschnitt sollen einige Teile der Makro-Sprache erläutert werden, die in Base – vor allem bei Formularen – immer wieder benutzt werden. (Soweit möglich und sinnvoll, werden dabei die Beispiele der folgenden Abschnitte benutzt.)

Der «Rahmen» eines Makros

Die Definition eines Makros beginnt mit dem Typ des Makros – **SUB** oder **FUNCTION** – und endet mit **END SUB** bzw. **END FUNCTION**. Einem Makro, das einem Ereignis zugewiesen wird, können Argumente (Werte) übergeben werden; sinnvoll ist aber nur das Argument **oEvent**. Alle anderen Routinen, die von einem solchen Makro aufgerufen werden, können abhängig vom Zweck mit oder ohne Rückgabewert definiert werden und beliebig mit Argumenten versehen werden.

```
SUB Ausleihe_aktualisieren
END SUB

SUB Zu_Formular_von_Formular(oEvent AS OBJECT)
END SUB

FUNCTION Loeschen_bestaetigen(oEvent AS OBJECT) AS BOOLEAN
    Loeschen_bestaetigen = FALSE
END FUNCTION
```

Es ist hilfreich, diesen Rahmen sofort zu schreiben und den Inhalt anschließend einzufügen. Bitte vergessen Sie nicht, Kommentare zur Bedeutung des Makros nach dem Grundsatz «so viel wie nötig, so wenig wie möglich» einzufügen. Außerdem unterscheidet Basic nicht zwischen Groß- und Kleinschreibung. In der Praxis werden feststehende Begriffe wie **SUB** vorzugsweise groß geschrieben, während andere Bezeichner Groß- und Kleinbuchstaben mischen.

Variablen definieren

Im nächsten Schritt werden – am Anfang der Routine – mit der **DIM**-Anweisung die Variablen, die innerhalb der Routine vorkommen, mit dem jeweiligen Datentyp definiert. Basic selbst verlangt das nicht, sondern akzeptiert, dass während des Programmablaufs neue Variablen auftreten. Der Programmcode ist aber «sicherer», wenn die Variablen und vor allem die Datentypen festgelegt sind. Viele Programmierer verpflichten sich selbst dazu, indem sie Basic über **Option Explicit** gleich zu Beginn eines Moduls mitteilen: Erzeuge nicht automatisch irgendwelche Variablen, sondern nutze nur die, die ich auch vorher definiert habe.

```
DIM oDoc AS OBJECT
DIM oDrawpage AS OBJECT
DIM oForm AS OBJECT
DIM sName AS STRING
DIM bOKEnabled AS BOOLEAN
DIM iCounter AS INTEGER
DIM dBirthday AS DATE
```

Für die Namensvergabe stehen nur Buchstaben (A–Z oder a–z), Ziffern und der Unterstrich '_' zur Verfügung, aber keine Umlaute oder Sonderzeichen. (Unter Umständen ist das Leerzeichen zulässig. Sie sollten aber besser darauf verzichten.) Das erste Zeichen muss ein Buchstabe sein.

Üblich ist es, durch den ersten Buchstaben den Datentyp deutlich zu machen.[1] Dann erkennt man auch mitten im Code den Typ der Variablen. Außerdem sind «sprechende Bezeichner» zu empfehlen, sodass die Bedeutung der Variablen schon durch den Namen erkannt werden kann.

1 Die Kennzeichnung sollte eventuell noch verfeinert werden, da mit nur einem Buchstaben zwischen dem Datentyp «Double» und dem Datentyp «Date» bzw. «Single» und «String» nicht unterschieden werden kann.

Die Liste der möglichen Datentypen in Star-Basic steht im Anhang des Handbuches. An verschiedenen Stellen sind Unterschiede zwischen der Datenbank, von Basic und der LibreOffice-API zu beachten. Darauf wird bei den Beispielen hingewiesen.

Arrays definieren

Gerade für Datenbanken ist die Sammlung von mehreren Variablen in einem Datensatz von Bedeutung. Werden mehrere Variablen zusammen in einer gemeinsamen Variablen gespeichert, so wird dies als ein Array bezeichnet. Ein Array muss definiert werden, bevor Daten in das Array geschrieben werden können.

```
DIM arDaten()
```

erzeugt eine leeres Array.

```
arDaten = array("Lisa","Schmidt")
```

So wird ein Array auf eine bestimmte Größe von 2 Elementen festgelegt und gleichzeitig mit Daten versehen.

Über

```
print arDaten(0), arDaten(1)
```

werden die beiden definierten Inhalte auf dem Bildschirm ausgegeben. Die Zählung für die Felder beginnt hier mit 0.

```
DIM arDaten(2)
arDaten(0) = "Lisa"
arDaten(1) = "Schmidt"
arDaten(2) = "Köln"
```

Dies erstellt ein Array, in dem 3 Elemente beliebigen Typs gespeichert werden können, also z.B. ein Datensatz mit den Variablen «Lisa» «Schmidt» «Köln». Mehr passt leider in dieses Array nicht hinein. Sollen mehr Elemente gespeichert werden, so muss das Array vergrößert werden. Wird während der Laufzeit eines Makros allerdings die Größe eines Arrays einfach nur neu definiert, so ist das Array anschließend leer wie eben ein neues Array.

```
ReDIM Preserve arDaten(3)
arDaten(3) = "18.07.2003"
```

Durch den Zusatz **`Preserve`** werden die vorherigen Daten beibehalten, das Array also tatsächlich zusätzlich um die Datumseingabe, hier in Form eines Textes, erweitert.

Das oben aufgeführte Array kann leider nur einen einzelnen Satz an Daten speichern. Sollen stattdessen, wie in einer Tabelle einer Datenbank, mehrere Datensätze gespeichert werden, so muss dem Array zu Beginn eine zusätzliche Dimension hinzugefügt werden.

```
DIM arDaten(2,1)
arDaten(0,0) = "Lisa"
arDaten(1,0) = "Schmidt"
arDaten(2,0) = "Köln"
arDaten(0,1) = "Egon"
arDaten(1,1) = "Müller"
arDaten(2,1) = "Hamburg"
```

Auch hier gilt bei einer Erweiterung über das vorher definierte Maß hinaus, dass der Zusatz **`Preserve`** die vorher eingegebenen Daten mit übernimmt.

Zugriff auf das Formular

Das Formular liegt in dem momentan aktiven Dokument. Der Bereich, der dargestellt wird, wird als **`drawpage`** bezeichnet. Der Behälter, in dem alle Formulare aufbewahrt werden, heißt **`forms`** – im Formularnavigator ist dies sozusagen der oberste Begriff, an den dann sämtliche Formulare angehängt werden. Die o.g. Variablen erhalten auf diesem Weg ihre Werte:

```
oDoc = thisComponent
oDrawpage = oDoc.drawpage
```

```
oForm = oDrawpage.forms.getByName("Filter")
```

Das Formular, auf das zugegriffen werden soll, ist hier mit dem Namen "Filter" versehen. Dies ist der Name, der auch im Formularnavigator in der obersten Ebene sichtbar ist. (Standardmäßig erhält das erste Formular den Namen "MainForm".) Unterformulare liegen – hierarchisch angeordnet – innerhalb eines Formulars und können Schritt für Schritt erreicht werden:

```
DIM oSubForm AS OBJECT
DIM oSubSubForm AS OBJECT
oSubForm = oForm.getByName("Leserauswahl")
oSubSubForm = oSubForm.getByName("Leseranzeige")
```

Anstelle der Variablen in den «Zwischenstufen» kann man auch direkt zu einem bestimmten Formular gelangen. Ein Objekt der Zwischenstufen, das mehr als einmal verwendet wird, sollte selbständig deklariert und zugewiesen werden. (Im folgenden Beispiel wird **`oSubForm`** nicht mehr benutzt.)

```
oForm = thisComponent.drawpage.forms.getByName("Filter")
oSubSubForm = oForm.getByName("Leserauswahl").getByName("Leseranzeige")
```

Hinweis

Sofern ein Name ausschließlich aus Buchstaben und Ziffern besteht (keine Umlaute, keine Leer- oder Sonderzeichen), kann der Name in der Zuweisung auch direkt verwendet werden:

```
oForm = thisComponent.drawpage.forms.Filter
oSubSubForm = oForm.Leserauswahl.Leseranzeige
```

Anders als bei Basic sonst üblich, ist bei solchen Namen auf Groß- und Kleinschreibung genau zu achten.

Einen anderen Zugang zum Formular ermöglicht das auslösende Ereignis für das Makro.

Startet ein Makro über ein Ereignis des Formulars, wie z. B. **Formular-Eigenschaften → Vor der Datensatzaktion**, so wird das Formular selbst folgendermaßen erreicht:

```
SUB MakrobeispielBerechne(oEvent AS OBJECT)
    oForm = oEvent.Source
    ...
END SUB
```

Startet ein Makro über ein Ereignis eines Formularfeldes, wie z. B. Eigenschaften: **Textfeld → Bei Fokusverlust**, so kann sowohl das Formularfeld als auch das Formular ermittelt werden:

```
SUB MakrobeispielBerechne(oEvent AS OBJECT)
    oFeld = oEvent.Source.Model
    oForm = oFeld.Parent
    ...
END SUB
```

Die Zugriffe über das Ereignis haben den Vorteil, dass kein Gedanke darüber verschwendet werden muss, ob es sich bei dem Formular um ein Hauptformular oder Unterformular handelt. Auch interessiert der Name des Formulars für die Funktionsweise des Makros nicht.

Zugriff auf Elemente eines Formulars

In gleicher Weise kann man auf die Elemente eines Formulars zugreifen: Deklarieren Sie eine entsprechende Variable als **`object`** und suchen Sie das betreffende Kontrollfeld innerhalb des Formulars:

```
DIM btnOK AS OBJECT  ' Button »OK»
btnOK = oSubSubForm.getByName("Schaltfläche 1")  ' aus dem Formular Leseranzeige
```

Dieser Weg funktioniert immer dann, wenn bekannt ist, mit welchem Element das Makro arbeiten soll. Wenn aber im ersten Schritt zu prüfen ist, welches Ereignis das Makro gestartet hat, ist der o.g. Weg über **`oEvent`** sinnvoll. Dann wird die Variable innerhalb des Makro-"Rahmens" deklariert

und beim Start des Makros zugewiesen. Die Eigenschaft **Source** liefert immer dasjenige Element, das das Makro gestartet hat; die Eigenschaft **Model** beschreibt das Kontrollfeld im Einzelnen:

```
SUB Auswahl_bestaetigen(oEvent AS OBJECT)
    DIM btnOK AS OBJECT
    btnOK = oEvent.Source.Model
END
```

Mit dem Objekt, das man auf diesem Weg erhält, werden die weiteren angestrebten Maßnahmen ausgeführt.

Bitte beachten Sie, dass auch Unterformulare als Bestandteile eines Formulars gelten.

Zugriff auf die Datenbank

Normalerweise wird der Zugriff auf die Datenbank über Formulare, Abfragen, Berichte oder die Serienbrief-Funktion geregelt, wie es in allen vorhergehenden Kapiteln beschrieben wurde. Wenn diese Möglichkeiten nicht genügen, kann ein Makro auch gezielt die Datenbank ansprechen, wofür es mehrere Wege gibt.

Die Verbindung zur Datenbank

Das einfachste Verfahren benutzt dieselbe Verbindung wie das Formular, wobei **oForm** wie oben bestimmt wird:

```
DIM oConnection AS OBJECT
oConnection = oForm.activeConnection()
```

Oder man holt die Datenquelle, also die Datenbank, durch das Dokument und benutzt die vorhandene Verbindung auch für das Makro:

```
DIM oDatasource AS OBJECT
DIM oConnection AS OBJECT
oDatasource = thisComponent.Parent.dataSource
oConnection = oDatasource.getConnection("","")
```

Ein weiterer Weg stellt sicher, dass bei Bedarf die Verbindung zur Datenbank hergestellt wird:

```
DIM oDatasource AS OBJECT
DIM oConnection AS OBJECT
oDatasource = thisComponent.Parent.CurrentController
IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
oConnection = oDatasource.ActiveConnection()
```

Die **IF**-Bedingung bezieht sich hier nur auf eine Zeile. Deshalb ist **END IF** nicht erforderlich.

Wenn das Makro durch die Benutzeroberfläche – nicht aus einem Formulardokument heraus – gestartet werden soll, ist folgende Variante geeignet:

```
DIM oDatasource AS OBJECT
DIM oConnection AS OBJECT
oDatasource = thisDatabaseDocument.CurrentController
IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
oConnection = oDatasource.ActiveConnection()
```

Der Zugriff auf Datenbanken außerhalb der aktuellen Datenbank ist folgendermaßen möglich:

```
DIM oDatabaseContext AS OBJECT
DIM oDatasource AS OBJECT
DIM oConnection AS OBJECT
oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
oDatasource = oDatabaseContext.getByName("angemeldeter Name der Datenbank in LO")
oConnection = oDatasource.GetConnection("","")
```

Auch die Verbindung zu nicht in LO angemeldete Datenbanken ist möglich. Hier muss dann lediglich statt des angemeldeten Namens der Pfad zur Datenbank mit «file:///..../Datenbank.odb» angegeben werden.

Ergänzende Hinweise zur Datenbankverbindung stehen im Abschnitt «*Verbindung mit Datenbanken erzeugen*».

SQL-Befehle

Die Arbeit mit der Datenbank erfolgt über SQL-Befehle. Ein solcher muss also erstellt und an die Datenbank geschickt werden; je nach Art des Befehls wird das Ergebnis ausgewertet und weiter verarbeitet. Mit der Anweisung **`createStatement`** wird das Objekt dafür erzeugt:

```
DIM oSQL_Statement AS OBJECT    ' das Objekt, das den SQL-Befehl ausführt
DIM stSql AS STRING             ' Text des eigentlichen SQL-Befehls
DIM oResult AS OBJECT           ' Ergebnis für executeQuery
DIM iResult AS INTEGER          ' Ergebnis für executeUpdate
oSQL_Statement = oConnection.createStatement()
```

Um *Daten abzufragen,* wird mit dem Befehl die Methode **`executeQuery`** aufgerufen und ausgeführt; das Ergebnis wird anschließend ausgewertet. Tabellennamen und Feldnamen werden üblicherweise in doppelte Anführungszeichen gesetzt. Diese müssen im Makro durch weitere doppelte Anführungszeichen maskiert werden, damit sie im Befehl erscheinen.

```
stSql = "SELECT * FROM ""Tabelle1"""
oResult = oSQL_Statement.executeQuery(stSql)
```

Um *Daten zu ändern* – also für **`INSERT`**, **`UPDATE`** oder **`DELETE`** – oder um die *Struktur der Datenbank* zu beeinflussen, wird mit dem Befehl die Methode **`executeUpdate`** aufgerufen und ausgeführt. Je nach Art des Befehls und der Datenbank erhält man kein nutzbares Ergebnis (ausgedrückt durch die Zahl 0) oder die Anzahl der bearbeiteten Datensätze.

```
stSql = "DROP TABLE ""Suchtmp"" IF EXISTS"
iResult = oSQL_Statement.executeUpdate(stSql)
```

Der Vollständigkeit halber sei noch ein Spezialfall erwähnt: Wenn **`oSQL_Statement`** unterschiedlich für **`SELECT`** oder für andere Zwecke benutzt wird, steht die Methode **`execute`** zur Verfügung. Diese benutzen wir nicht; wir verweisen dazu auf die API-Referenz.

Vorbereitete SQL-Befehle mit Parametern

In allen Fällen, in denen manuelle Eingaben der Benutzer in einen SQL-Befehl übernommen werden, ist es einfacher und sicherer, den Befehl nicht als lange Zeichenkette zu erstellen, sondern ihn vorzubereiten und mit Parametern zu benutzen. Das vereinfacht die Formatierung von Zahlen, Datumsangaben und auch Zeichenketten (die ständigen doppelten Anführungszeichen entfallen) und verhindert Datenverlust durch böswillige Eingaben.

Bei diesem Verfahren wird zunächst das Objekt für einen bestimmten SQL-Befehl erstellt und vorbereitet:

```
DIM oSQL_Statement AS OBJECT    ' das Objekt, das den SQL-Befehl ausführt
DIM stSql AS STRING             ' Text des eigentlichen SQL-Befehls
stSql = "UPDATE Verfasser " _
      & "SET Nachname = ?, Vorname = ?" _
      & "WHERE ID = ?"
oSQL_Statement = oConnection.prepareStatement(stSql)
```

Das Objekt wird mit **`prepareStatement`** erzeugt, wobei der SQL-Befehl bereits bekannt sein muss. Jedes Fragezeichen markiert eine Stelle, an der später – vor der Ausführung des Befehls – ein konkreter Wert eingetragen wird. Durch das «Vorbereiten» des Befehls stellt sich die Datenbank darauf ein, welche Art von Angaben – in diesem Fall zwei Zeichenketten und eine Zahl – vorgesehen ist. Die verschiedenen Stellen werden durch die Position (ab 1 gezählt) unterschieden.

Anschließend werden mit passenden Anweisungen die Werte übergeben und danach der SQL-Befehl ausgeführt. Die Werte werden hier aus Kontrollfeldern des Formulars übernommen, können aber auch aus anderen Makro-Elementen stammen oder im Klartext angegeben werden:

```
oSQL_Statement.setString(1, oTextfeld1.Text)    ' Text für den Nachnamen
oSQL_Statement.setString(2, oTextfeld2.Text)    ' Text für den Vornamen
oSQL_Statement.setLong(3, oZahlenfeld1.Value)   ' Wert für die betreffende ID
```

```
iResult = oSQL_Statement.executeUpdate
```

Die vollständige Liste der Zuweisungen findet sich im Abschnitt «*Parameter für vorbereitete SQL-Befehle*».

Wer sich weiter über die Vorteile dieses Verfahrens informieren möchte, findet hier Erläuterungen:

- *SQL-Injection (Wikipedia)* (http://de.wikipedia.org/wiki/SQL-Injection)
- *Why use PreparedStatement (Java JDBC)* (http://javarevisited.blogspot.de/2012/03/why-use-preparedstatement-in-java-jdbc.html)
- *SQL-Befehle (Einführung in SQL)* (http://de.wikibooks.org/wiki/Einführung_in_SQL:_SQL-Befehle#Hinweis_f.C3.BCr_Programmierer:_Parameter_benutzen.21)

Datensätze lesen und benutzen

Es gibt – abhängig vom Zweck – mehrere Wege, um Informationen aus einer Datenbank in ein Makro zu übernehmen und weiter zu verarbeiten.

Bitte beachten Sie: Wenn hier von einem «Formular» gesprochen wird, kann es sich auch um ein Unterformular handeln. Es geht dann immer über dasjenige (Teil-) Formular, das mit einer bestimmten Datenmenge verbunden ist.

Mithilfe des Formulars

Der aktuelle Datensatz und seine Daten stehen immer über das Formular zur Verfügung, das die betreffende Datenmenge (Tabelle, Abfrage, SELECT) anzeigt. Dafür gibt es mehrere Methoden, die mit **`get`** und dem Datentyp bezeichnet sind, beispielsweise diese:

```
DIM ID AS LONG
DIM sName AS STRING
DIM dValue AS CURRENCY
DIM dEintritt AS NEW com.sun.star.util.Date
ID = oForm.getLong(1)
sName = oForm.getString(2)
dValue = oForm.getDouble(4)
dEintritt = oForm.getDate(7)
```

Bei allen diesen Methoden ist jeweils die Nummer der Spalte in der Datenmenge anzugeben – gezählt ab 1.

Hinweis	Bei allen Methoden, die mit Datenbanken arbeiten, wird ab 1 gezählt. Das gilt sowohl für Spalten als auch für Zeilen.

Möchte man anstelle der Spaltennummern mit den Spaltennamen der zugrundeliegenden Datenmenge (Tabelle, Abfrage, Ansicht (*View*)) arbeiten, kann man die Spaltennummer über die Methode **`findColumn`** ermitteln – hier ein Beispiel zum Auffinden der Spalte "Name":

```
DIM sName AS STRING
nName = oForm.findColumn("Name")
sName = oForm.getString(nName)
```

Man erhält immer einen Wert des Typs der Methode, wobei die folgenden Sonderfälle zu beachten sind.

- Es gibt keine Methode für Daten des Typs **`Decimal`**, **`Currency`** o.ä., also für kaufmännisch exakte Berechnungen. Da Basic automatisch die passende Konvertierung vornimmt, kann ersatzweise **`getDouble`** verwendet werden.
- Bei **`getBoolean`** ist zu beachten, wie in der Datenbank «Wahr» und «Falsch» definiert sind. Die «üblichen» Definitionen (logische Werte, 1 als «Wahr») werden richtig verarbeitet.
- Datumsangaben können nicht nur mit dem Datentyp **`DATE`** definiert werden, sondern auch (wie oben) als **`util.Date`**. Das erleichtert u.a. Lesen und Ändern von Jahr, Monat, Tag.

- Bei ganzen Zahlen sind Unterschiede der Datentypen zu beachten. Im obigen Beispiel wird **`getLong`** verwendet; auch die Basic-Variable ID muss den Datentyp **`Long`** erhalten, da dieser vom Umfang her mit **`Integer`** aus der Datenbank übereinstimmt.

Die vollständige Liste dieser Methoden findet sich im Abschnitt «*Datenzeilen bearbeiten*».

Tipp

Sollen Werte aus einem Formular für direkte Weiterverarbeitung in SQL genutzt werden (z.B. für die Eingabe der Daten in eine andere Tabelle), so ist es wesentlich einfacher, nicht nach dem Typ der Felder zu fragen.

Das folgende Makro, an **Eigenschaften: Schaltfläche → Ereignisse → Aktion ausführen** gekoppelt, liest das erste Feld des Formulars aus – unabhängig von dem für die Weiterverarbeitung in Basic erforderlichen Typ.

```
SUB WerteAuslesen(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    DIM stFeld1 AS STRING
    oForm = oEvent.Source.Model.Parent
    stFeld1 = oForm.getString(1)
END SUB
```

Werden die Felder über **`getString()`** ausgelesen, so werden die Formatierungen beibehalten, die für eine Weiterverarbeitung in SQL notwendig sind. Ein Datum, das in einem deutschsprachigen Formular als '08.03.15' dargestellt wird, wird so im Format '2015-03-08' ausgelesen und kann direkt in SQL weiter verarbeitet werden.

Die Auslesung in dem dem Typ entsprechenden Format ist nur erforderlich, wenn im Makro Werte weiter verarbeitet, z.B. mit ihnen gerechnet werden soll.

Ergebnis einer Abfrage

In gleicher Weise kann die Ergebnismenge einer Abfrage benutzt werden. Im Abschnitt «*SQL-Befehle*» steht die Variable **`oResult`** für diese Ergebnismenge, die üblicherweise so oder ähnlich ausgelesen wird:

```
WHILE oResult.next          ' einen Datensatz nach dem anderen verarbeiten
    rem übernimm die benötigten Werte in Variablen
    stVar = oResult.getString(1)
    inVar = oResult.getLong(2)
    boVar = oResult.getBoolean(3)
    rem mach etwas mit diesen Werten
WEND
```

Je nach Art des SQL-Befehls, dem erwarteten Ergebnis und dem Zweck kann vor allem die **`WHILE`**-Schleife verkürzt werden oder sogar entfallen. Aber grundsätzlich wird eine Ergebnismenge immer nach diesem Schema ausgewertet.

Soll nur der erste Datensatz ausgewertet werden, so wird mit

```
oResult.next
```

zuerst die Zeile auf diesen Datensatz bewegt und dann mit

```
stVar = oResult.getString(1)
```

z.B. der Inhalt des ersten Datenfeldes gelesen. Die Schleife entfällt hier.

Die Abfrage zu dem obigen Beispiel hat in der ersten Spalte einen Text, in der zweiten Spalte einen Integer-Zahlenwert (**`Integer`** aus der Datenbank entspricht **`Long`** in Basic) und in der dritten Spalte ein Ja/Nein-Feld. Die Felder werden durch den entsprechenden Indexwert angesprochen. Der Index für die Felder beginnt hier, im Gegensatz zu der sonstigen Zählung bei Arrays, mit dem Wert '1'.

In dem so erstellten Ergebnis ist allerdings keine Navigation möglich. Nur einzelne Schritte zum nächsten Datensatz sind erlaubt. Um innerhalb der Datensätze navigieren zu können, muss der **`ResultSetType`** bei der Erstellung der Abfrage bekannt sein. Hierauf wird über

```
oSQL_Anweisung.ResultSetType = 1004
```

oder

```
oSQL_Anweisung.ResultSetType = 1005
```

zugegriffen. Der Typ **1004 - SCROLL_INTENSIVE** erlaubt eine beliebige Navigation. Allerdings bleibt eine Änderung an den Originaldaten während des Auslesens unbemerkt. Der Typ **1005 – SCROLL_SENSITIVE** berücksichtigt zusätzlich gegebenenfalls Änderungen an den Originaldaten, die das Abfrageergebnis beeinflussen könnten.

Soll zusätzlich in dem Ergebnissatz eine Änderung der Daten ermöglicht werden, so muss die **ResultSetConcurrency** vorher definiert werden. Die Update-Möglichkeit wird über

```
oSQL_Anweisung.ResultSetConcurrency = 1008
```

hergestellt. Der Typ **1007 - READ_ONLY** ist hier die Standardeinstellung.

Die Anzahl der Zeilen, die die Ergebnismenge enthält, kann nur nach Wahl der entsprechenden Typen so bestimmt werden:

```
DIM iResult AS LONG
IF oResult.last THEN           ' gehe zum letzten Datensatz, sofern möglich
   iResult = oResult.getRow    ' die laufende Nummer ist die Anzahl
ELSE
   iResult = 0
END IF
```

Mithilfe eines Kontrollfelds

Wenn ein Kontrollfeld mit einer Datenmenge verbunden ist, kann der Wert auch direkt ausgelesen werden, wie es im nächsten Abschnitt beschrieben wird. Das ist aber teilweise mit Problemen verbunden. Sicherer ist – neben dem Verfahren «*Mithilfe des Formulars*» – der folgende Weg, der für verschiedene Kontrollfelder gezeigt wird:

```
sValue = oTextField.BoundField.Text        ' Beispiel für ein Textfeld
nValue = oNumericField.BoundField.Value    ' Beispiel für ein numerisches Feld
dValue = oDateField.BoundField.Date        ' Beispiel für ein Datumsfeld
```

BoundField stellt dabei die Verbindung her zwischen dem (sichtbaren) Kontrollfeld und dem eigentlichen Inhalt der Datenmenge.

Datensätze wechseln und bestimmte Datensätze ansteuern

Im vorletzten Beispiel wurde mit der Methode **Next** von einer Zeile der Ergebnismenge zur nächsten gegangen. In gleicher Weise gibt es weitere Maßnahmen und Prüfungen, und zwar sowohl für die Daten eines Formulars – angedeutet durch die Variable **oForm** – als auch für eine Ergebnismenge. Beispielsweise kann man beim Verfahren «*Automatisches Aktualisieren von Formularen*» den vorher aktuellen Datensatz wieder markieren:

```
DIM loRow AS LONG
loRow = oForm.getRow()    ' notiere die aktuelle Zeilennummer
oForm.reload()           ' lade die Datenmenge neu
oForm.absolute(loRow)     ' gehe wieder zu der notierten Zeilennummer
```

Im Abschnitt «*In einer Datenmenge navigieren*» stehen alle dazu passenden Methoden.

Datensätze bearbeiten – neu anlegen, ändern, löschen

Um Datensätze zu bearbeiten, müssen mehrere Teile zusammenpassen: Eine Information muss vom Anwender in das Kontrollfeld gebracht werden; das geschieht durch die Tastatureingabe. Anschließend muss die Datenmenge «dahinter» diese Änderung zur Kenntnis nehmen; das geschieht durch das Verlassen eines Feldes und den Wechsel zum nächsten Feld. Und schließlich muss die Datenbank selbst die Änderung erfahren; das erfolgt durch den Wechsel von einem Datensatz zu einem anderen.

Bei der Arbeit mit einem Makro müssen ebenfalls diese Teilschritte beachtet werden. Wenn einer fehlt oder falsch ausgeführt wird, gehen Änderungen verloren und «landen» nicht in der Datenbank. In erster Linie muss die Änderung nicht in der Anzeige des Kontrollfelds erscheinen, sondern in der Datenmenge. Es ist deshalb sinnlos, die Eigenschaft **`Text`** des Kontrollfelds zu ändern.

Bitte beachten Sie, dass nur Datenmengen vom Typ «Tabelle» problemlos geändert werden können. Bei anderen Datenmengen ist dies nur unter besonderen Bedingungen möglich.

Inhalt eines Kontrollfelds ändern

Wenn es um die Änderung eines einzelnen Wertes geht, wird das über die Eigenschaft **`BoundField`** des Kontrollfelds mit einer passenden Methode erledigt. Anschließend muss nur noch die Änderung an die Datenbank weitergegeben werden. Beispiel für ein Datumsfeld, in das das aktuelle Datum eingetragen werden soll:

```
DIM unoDate AS NEW com.sun.star.util.Date
unoDate.Year = Year(Date)
unoDate.Month = Month(Date)
unoDate.Day = Day(Date)
oDateField.BoundField.updateDate( unoDate )
oForm.updateRow()       ' Weitergabe der Änderung an die Datenbank
```

Für **`BoundField`** wird diejenige der **`updateXxx`**-Methoden aufgerufen, die zum Datentyp des Feldes passt – hier geht es um einen **`Date`**-Wert. Als Argument wird der gewünschte Wert übergeben – hier das aktuelle Datum, konvertiert in die vom Makro benötigte Schreibweise. Die entsprechende Erstellung des Datums kann auch durch die Formel **`CDateToUnoDate`** erreicht werden:

```
oDateField.BoundField.updateDate( CDateToUnoDate(NOW()) )
oForm.updateRow()       ' Weitergabe der Änderung an die Datenbank
```

Zeile einer Datenmenge ändern

Wenn mehrere Werte in einer Zeile geändert werden sollen, ist der vorstehende Weg ungeeignet. Zum einen müsste für jeden Wert ein Kontrollfeld existieren, was oft nicht gewünscht oder sinnvoll ist. Zum anderen muss man sich für jedes dieser Felder ein Objekt «holen». Der einfache und direkte Weg geht über das Formular, beispielsweise so:

```
DIM unoDate AS NEW com.sun.star.util.Date
unoDate.Year = Year(Date)
unoDate.Month = Month(Date)
unoDate.Day = Day(Date)
oForm.updateDate(3, unoDate )
oForm.updateString(4, "ein Text")
oForm.updateDouble(6, 3.14)
oForm.updateInt(7, 16)
oForm.updateRow()
```

Für jede Spalte der Datenmenge wird die zum Datentyp passende **`updateXxx`**-Methode aufgerufen. Als Argumente werden die Nummer der Spalte (ab 1 gezählt) und der jeweils gewünschte Wert übergeben. Anschließend muss nur noch die Änderung an die Datenbank weitergegeben werden.

Zeilen anlegen, ändern, löschen

Die genannten **Änderungen** beziehen sich immer auf die aktuelle Zeile der Datenmenge des Formulars. Unter Umständen muss vorher eine der Methoden aus «*In einer Datenmenge navigieren*» aufgerufen werden. Es werden also folgende Maßnahmen benötigt:

1. Wähle den aktuellen Datensatz.
2. Ändere die gewünschten Werte, wie im vorigen Abschnitt beschrieben.
3. Bestätige die Änderungen mit folgendem Befehl:
   ```
   oForm.updateRow()
   ```

4. Als Sonderfall ist es auch möglich, die Änderungen zu verwerfen und den vorherigen Zustand wiederherzustellen:
   ```
   oForm.cancelRowUpdates()
   ```

Für einen **neuen Datensatz** gibt es eine spezielle Methode (vergleichbar mit dem Wechsel in eine neue Zeile im Tabellenkontrollfeld). Es werden also folgende Maßnahmen benötigt:

1. Bereite einen neuen Datensatz vor:
   ```
   oForm.moveToInsertRow()
   ```
2. Trage alle vorgesehenen und benötigten Werte ein. Dies geht ebenfalls mit den **`updateXxx`**-Methoden, wie im vorigen Abschnitt beschrieben.
3. Bestätige die Neuaufnahme mit folgendem Befehl:
   ```
   oForm.insertRow()
   ```
4. Die Neuaufnahme kann nicht einfach rückgängig gemacht werden. Stattdessen ist die soeben neu angelegte Zeile wieder zu löschen.

Für das **Löschen** eines Datensatzes gibt es einen einfachen Befehl; es sind also folgende Maßnahmen nötig:

1. Wähle – wie für eine Änderung – den gewünschten Datensatz und mache ihn zum aktuellen.
2. Bestätige die Löschung mit folgendem Befehl:
   ```
   oForm.deleteRow()
   ```

Tipp

Damit eine Änderung in die Datenbank übernommen wird, ist sie durch **`updateRow`** bzw. **`insertRow`** ausdrücklich zu bestätigen. Während beim Betätigen des Speicher-Buttons die passende Funktion automatisch ermittelt wird, muss vor dem Abspeichern ermittelt werden, ob der Datensatz neu ist (**`Insert`**) oder ein bestehender Datensatz bearbeitet wurde (**`Update`**).

```
IF oForm.isNew THEN
   oForm.insertRow()
ELSE
   oForm.updateRow()
END IF
```

Kontrollfelder prüfen und ändern

Neben dem Inhalt, der aus der Datenmenge kommt, können viele weitere Informationen zu einem Kontrollfeld gelesen, verarbeitet und geändert werden. Das betrifft vor allem die Eigenschaften, die im Kapitel «Formulare» aufgeführt werden. Eine Übersicht steht im Abschnitt «*Eigenschaften bei Formularen und Kontrollfeldern*».

In mehreren Beispielen des Abschnitts «*Bedienbarkeit verbessern*» wird die Zusatzinformation eines Feldes benutzt:

```
DIM stTag AS STRING
stTag = oEvent.Source.Model.Tag
```

Die Eigenschaft **`Text`** kann – wie im vorigen Abschnitt erläutert – nur dann sinnvoll geändert werden, wenn das Feld nicht mit einer Datenmenge verbunden ist. Aber andere Eigenschaften, die «eigentlich» bei der Formulardefinition festgelegt werden, können zur Laufzeit angepasst werden. Beispielsweise kann in einem Beschriftungsfeld die Textfarbe gewechselt werden, wenn statt einer Meldung ein Hinweis oder eine Warnung angezeigt werden soll:

```
SUB showWarning(oField AS OBJECT, iType AS INTEGER)
   SELECT CASE iType
      CASE 1
         oField.TextColor = RGB(0,0,255)    ' 1 = blau
      CASE 2
         oField.TextColor = RGB(255,0,0)    ' 2 = rot
```

```
        CASE ELSE
            oField.TextColor = RGB(0,255,0)    ' 0 = grün (weder 1 noch 2)
    END SELECT
END SUB
```

Englische Bezeichner in Makros

Während der Formular-Designer in der deutschen Version auch deutsche Bezeichnungen für die Eigenschaften und den Datenzugriff verwendet, müssen in Basic englische Begriffe verwendet werden. Diese sind in den folgenden Übersichten aufgeführt.

Eigenschaften, die üblicherweise nur in der Formular-Definition festgelegt werden, stehen nicht in den Übersichten. Gleiches gilt für Methoden (Funktionen und Prozeduren), die nur selten verwendet werden oder für die kompliziertere Erklärungen nötig wären.

Die Übersichten nennen folgende Angaben:

- *Name* Bezeichnung der Eigenschaft oder Methode im Makro-Code
- *Datentyp* Einer der Datentypen von Basic
  Bei Funktionen ist der Typ des Rückgabewerts angegeben; bei Prozeduren entfällt diese Angabe.
- *L/S* Hinweis darauf, wie der Wert der Eigenschaft verwendet wird:
  - *L* nur Lesen
  - *S* nur Schreiben (Ändern)
  - *(L)* Lesen möglich, aber für weitere Verarbeitung ungeeignet
  - *(S)* Schreiben möglich, aber nicht sinnvoll
  - *L+S* geeignet für Lesen und Schreiben

Weitere Informationen finden sich vor allem in der *API-Referenz* mit Suche nach der englischen Bezeichnung des Kontrollfelds. Gut geeignet, um herauszufinden, welche Eigenschaften und Methoden denn eigentlich bei einem Element zur Verfügung stehen, ist auch das Tool *Xray* .

```
SUB Main(oEvent)
    Xray(oEvent)
END SUB
```

Hiermit wird die Erweiterung Xray aus dem Aufruf heraus gestartet.

Eigenschaften bei Formularen und Kontrollfeldern

Das «Modell» eines Kontrollfelds beschreibt seine Eigenschaften. Je nach Situation kann der Wert einer Eigenschaft nur gelesen und nur geändert werden. Die Reihenfolge orientiert sich an den Aufstellungen «Eigenschaften der Kontrollfelder» im Kapitel «*Formular*».

Schrift

In jedem Kontrollfeld, das Text anzeigt, können die Eigenschaften der Schrift angepasst werden.

Name	*Datentyp*	*L/S*	*Eigenschaft*
FontName	string	L+S	Schriftart.
FontHeight	single	L+S	Schriftgröße.
FontWeight	single	L+S	Schriftstärke.
FontSlant	integer	L+S	Art der Schrägstellung.
FontUnderline	integer	L+S	Art der Unterstreichung.
FontStrikeout	integer	L+S	Art des Durchstreichens.

Formular

Englische Bezeichnung: *Form*

Name	Datentyp	L/S	Eigenschaft
ApplyFilter	boolean	L+S	Filter aktiviert.
Filter	string	L+S	Aktueller Filter für die Datensätze.
FetchSize	long	L+S	Anzahl der Datensätze, die «am Stück» geladen werden.
Row	long	L	Nummer der aktuellen Zeile.
RowCount	long	L	Anzahl der Datensätze. Entspricht der Anzeige der Gesamtdatensätze in der Navigationsleiste. Da nicht direkt alle Datensätze über FetchSize in den Cache gelesen werden steht hier z.B. '41*', obwohl die Tabelle deutlich mehr Datensätze hat. RowCount gibt dann leider auch nur '41' aus.

Einheitlich für alle Arten von Kontrollfeld

Englische Bezeichnung: *Control* – siehe auch *FormComponent*

Name	Datentyp	L/S	Eigenschaft
Name	string	L+(S)	Bezeichnung für das Feld.
Enabled	boolean	L+S	Aktiviert: Feld kann ausgewählt werden.
EnableVisible	boolean	L+S	Sichtbar: Feld wird dargestellt.
ReadOnly	boolean	L+S	Nur lesen: Inhalt kann nicht geändert werden.
TabStop	boolean	L+S	Feld ist in der Tabulator-Reihenfolge erreichbar.
Align	integer	L+S	Horizontale Ausrichtung: 0 = links, 1 = zentriert, 2 = rechts
BackgroundColor	long	L+S	Hintergrundfarbe.
Tag	string	L+S	Zusatzinformation.
HelpText	string	L+S	Hilfetext als «Tooltip».

Einheitlich für viele Arten von Kontrollfeld

Name	Datentyp	L/S	Eigenschaft
Text	string	(L+S)	Inhalt des Feldes aus der Anzeige. Bei Textfeldern nach dem Lesen auch zur weiteren Verarbeitung geeignet, andernfalls nur in Ausnahmefällen.
Spin	boolean	L+S	Drehfeld eingeblendet (bei formatierten Feldern).
TextColor	long	L+S	Textfarbe.
DataField	string	L	Name des Feldes aus der Datenmenge
BoundField	object	L	Objekt, das die Verbindung zur Datenmenge herstellt und vor allem dem Zugriff auf den Feldinhalt dient.

Textfeld – weitere Angaben

Englische Bezeichnung: *TextField*

Name	Datentyp	L/S	Eigenschaft
String	string	L+S	Inhalt des Feldes aus der Anzeige.

Name	Datentyp	L/S	Eigenschaft
MaxTextLen	integer	L+S	Maximale Textlänge.
DefaultText	string	L+S	Standardtext.
MultiLine	boolean	L+S	Mehrzeilig oder einzeilig.
EchoChar	(integer)	L+S	Zeichen für Kennwörter (Passwort-Eingabe verstecken).

Numerisches Feld

Englische Bezeichnung: *NumericField*

Name	Datentyp	L/S	Eigenschaft
ValueMin	double	L+S	Minimalwert zur Eingabe.
ValueMax	double	L+S	Maximalwert zur Eingabe.
Value	double	L+(S)	Aktueller Wert nicht für Werte aus der Datenmenge verwenden.
ValueStep	double	L+S	Intervall bei Verwendung mit Mausrad oder Drehfeld.
DefaultValue	double	L+S	Standardwert.
DecimalAccuracy	integer	L+S	Nachkommastellen.
ShowThousandsSeparator	boolean	L+S	Tausender-Trennzeichen anzeigen.

Datumsfeld

Englische Bezeichnung: *DateField*

Datumswerte werden als Datentyp **`long`** definiert und im ISO-Format YYYYMMDD angezeigt, also 20120304 für den 04.03.2012. Zur Verwendung dieses Typs zusammen mit **`getDate`** und **`updateDate`** sowie dem Typ **`com.sun.star.util.Date`** verweisen wir auf die Beispiele.

Name	Datentyp	Datentyp ab LO 4.1.1	L/S	Eigenschaft
DateMin	long	com.sun.star.util.Date	L+S	Minimalwert zur Eingabe.
DateMax	long	com.sun.star.util.Date	L+S	Maximalwert zur Eingabe.
Date	long	com.sun.star.util.Date	L+(S)	Aktueller Wert nicht für Werte aus der Datenmenge verwenden.
DateFormat	integer		L+S	Datumsformat nach Festlegung des Betriebssystems: 0 = kurze Datumsangabe (einfach) 1 = kurze Datumsangabe `tt.mm.jj` (Jahr zweistellig) 2 = kurze Datumsangabe `tt.mm.jjjj` (Jahr vierstellig) 3 = lange Datumsangabe (mit Wochentag und Monatsnamen) Weitere Möglichkeiten sind der Formulardefi-

Name	Daten-typ	Datentyp ab LO 4.1.1	L/S	Eigenschaft
				nition oder der *API-Referenz* zu entnehmen.
DefaultDate	long	com.sun.star .util.Date	L+S	Standardwert.
DropDown	boolean		L+S	Aufklappbaren Monatskalender anzeigen.

Zeitfeld

Englische Bezeichnung: *TimeField*

Auch Zeitwerte werden als Datentyp **`long`** definiert.

Name	Daten-typ	Datentyp ab LO 4.1.1	L/S	Eigenschaft
TimeMin	long	com.sun.star .util.Time	L+S	Minimalwert zur Eingabe.
TimeMax	long	com.sun.star .util.Time	L+S	Maximalwert zur Eingabe.
Time	long	com.sun.star .util.Time	L+(S)	Aktueller Wert nicht für Werte aus der Datenmenge verwenden.
TimeFormat	integer		L+S	Zeitformat: 0 = kurz als `hh:mm` (Stunde, Minute, 24 Stunden) 1 = lang als `hh:mm:ss` (dazu Sekunden, 24 Stunden) 2 = kurz als `hh:mm` (12 Stunden AM/PM) 3 = lang als `hh:mm:ss` (12 Stunden AM/PM) 4 = als kurze Angabe einer Dauer 5 = als lange Angabe einer Dauer
DefaultTime	long	com.sun.star .util.Time	L+S	Standardwert.

Währungsfeld

Englische Bezeichnung: *CurrencyField*

Ein Währungsfeld ist ein numerisches Feld mit den folgenden zusätzlichen Möglichkeiten.

Name	Datentyp	L/S	Eigenschaft
CurrencySymbol	string	L+S	Währungssymbol (nur zur Anzeige).
PrependCurrencySymbol	boolean	L+S	Anzeige des Symbols vor der Zahl.

Formatiertes Feld

Englische Bezeichnung: *FormattedControl*

Ein formatiertes Feld wird wahlweise für Zahlen, Währungen oder Datum/Zeit verwendet. Sehr viele der bisher genannten Eigenschaften gibt es auch hier, aber mit anderer Bezeichnung.

Name	Datentyp	L/S	Eigenschaft
CurrentValue	variant	L	Aktueller Wert des Inhalts; der konkrete Datentyp hängt vom Inhalt des Feldes und dem For-
EffectiveValue		L+(S)	

Name	Datentyp	L/S	Eigenschaft
			mat ab.
EffectiveMin	double	L+S	Minimalwert zur Eingabe.
EffectiveMax	double	L+S	Maximalwert zur Eingabe.
EffectiveDefault	variant	L+S	Standardwert.
FormatKey	long	L+(S)	Format für Anzeige und Eingabe. Es gibt kein einfaches Verfahren, das Format durch ein Makro zu ändern.
EnforceFormat	boolean	L+S	Formatüberprüfung: Bereits während der Eingabe sind nur zulässige Zeichen und Kombinationen möglich.

Listenfeld

Englische Bezeichnung: *ListBox*

Der Lese- und Schreibzugriff auf den Wert, der hinter der ausgewählten Zeile steht, ist etwas umständlich, aber möglich.

Name	Datentyp	L/S	Eigenschaft
ListSource	array of string	L+S	Datenquelle: Herkunft der Listeneinträge oder Name der Datenmenge, die die Einträge liefert.
ListSourceType	integer	L+S	Art der Datenquelle: 0 = Werteliste 1 = Tabelle 2 = Abfrage 3 = Ergebnismenge eines SQL-Befehls 4 = Ergebnis eines Datenbank-Befehls 5 = Feldnamen einer Datenbank-Tabelle
StringItemList	array of string	L	Listeneinträge, die zur Auswahl zur Verfügung stehen.
ItemCount	integer	L	Anzahl der vorhandenen Listeneinträge.
ValueItemList	array of string	L	Liste der Werte, die über das Formular an die Tabelle weitergegeben werden.
DropDown	boolean	L+S	Aufklappbar.
LineCount	integer	L+S	Anzahl der angezeigten Zeilen im aufgeklappten Zustand.
MultiSelection	boolean	L+S	Mehrfachselektion vorgesehen.
SelectedItems	array of integer	L+S	Liste der ausgewählten Einträge, und zwar als Liste der Positionen in der Liste aller Einträge.

Das (erste) ausgewählte Element aus dem Listenfeld erhält man auf diesem Weg:

```
oControl = oForm.getByName("Name des Listenfelds")
sEintrag = oControl.ValueItemList( oControl.SelectedItems(0) )
```

Hinweis

Seit LO 4.1 wird direkt der Wert ermittelt, der bei einem Listenfeld an die Datenbank weitergegeben wird.

```
oControl = oForm.getByName("Name des Listenfelds")
iD = oControl.getCurrentValue()
```

Mit `getCurrentValue()` wird also immer der Wert ausgegeben, der auch tatsächlich in der Tabelle der Datenbank abgespeichert wird. Dies ist beim Listenfeld von dem hiermit verknüpften gebundenen Feld (`BoundField`) abhängig.

Bis einschließlich LO 4.0 wurde hier immer der angezeigte Inhalt, nicht aber der an die darunterliegende Tabelle weitergegebene Wert wiedergegeben.

Soll für die Einschränkung einer Auswahlmöglichkeit die Abfrage für ein Listenfeld ausgetauscht werden, so ist dabei zu beachten, dass es sich bei dem Eintrag um ein «array of string» handelt:

```
SUB Listenfeldfilter
    DIM stSql(0) AS STRING
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("MainForm")
    oFeld = oForm.getByname("Listenfeld")
    stSql(0) = "SELECT ""Name"", ""ID"" FROM ""Filter_Name"" ORDER BY ""Name"""
    oFeld.ListSource = stSql
    oFeld.refresh
END SUB
```

Kombinationsfeld

Englische Bezeichnung: *ComboBox*

Trotz ähnlicher Funktionalität wie beim Listenfeld weichen die Eigenschaften teilweise ab.

Hier verweisen wir ergänzend auf das Beispiel «*Kombinationsfelder als Listenfelder mit Eingabemöglichkeit*».

Name	***Datentyp***	***L/S***	***Eigenschaft***
Autocomplete	boolean	L+S	Automatisch füllen.
StringItemList	array of string	L+S	Listeneinträge, die zur Auswahl zur Verfügung stehen.
ItemCount	integer	L	Anzahl der vorhandenen Listeneinträge.
DropDown	boolean	L+S	Aufklappbar.
LineCount	integer	L+S	Anzahl der angezeigten Zeilen im aufgeklappten Zustand.
Text	string	L+S	Aktuell angezeigter Text.
DefaultText	string	L+S	Standardeintrag.
ListSource	string	L+S	Name der Datenquelle, die die Listeneinträge liefert.
ListSourceType	integer	L+S	Art der Datenquelle; gleiche Möglichkeiten wie beim Listenfeld (nur die Auswahl «Werteliste» wird ignoriert).

Markierfeld, Optionsfeld

Englische Bezeichnungen: *CheckBox* (Markierfeld) bzw. *RadioButton* (Optionsfeld; auch «Option Button» möglich)

Name	Datentyp	L/S	Eigenschaft
Label	string	L+S	Titel (Beschriftung)
State	short	L+S	Status 0 = nicht ausgewählt 1 = ausgewählt 2 = unbestimmt
MultiLine	boolean	L+S	Wortumbruch (bei zu langem Text).
RefValue	string	L+S	Referenzwert

Maskiertes Feld

Englische Bezeichnung: *PatternField*

Neben den Eigenschaften für «einfache» Textfelder sind folgende interessant.

Name	Datentyp	L/S	Eigenschaft
EditMask	string	L+S	Eingabemaske.
LiteralMask	string	L+S	Zeichenmaske.
StrictFormat	boolean	L+S	Formatüberprüfung bereits während der Eingabe.

Tabellenkontrollfeld

Englische Bezeichnung: *GridControl*

Name	Datentyp	L/S	Eigenschaft
Count	long	L	Anzahl der Spalten.
ElementNames	array of string	L	Liste der Spaltennamen.
HasNavigationBar	boolean	L+S	Navigationsleiste vorhanden.
RowHeight	long	L+S	Zeilenhöhe.

Beschriftungsfeld

Englische Bezeichnung: *FixedText* – auch *Label* ist üblich

Name	Datentyp	L/S	Eigenschaft
Label	string	L+S	Der angezeigte Text.
MultiLine	boolean	L+S	Wortumbruch (bei zu langem Text).

Gruppierungsrahmen

Englische Bezeichnung: *GroupBox*

Keine Eigenschaft dieses Kontrollfelds wird üblicherweise durch Makros bearbeitet. Wichtig ist der Status der einzelnen Optionsfelder.

Schaltfläche

Englische Bezeichnungen: *CommandButton* – für die grafische Schaltfläche *ImageButton*

Name	Datentyp	L/S	Eigenschaft
Label	string	L+S	Titel – Text der Beschriftung.
State	short	L+S	Standardstatus «ausgewählt» bei «Umschalten».
MultiLine	boolean	L+S	Wortumbruch (bei zu langem Text).

Name	Datentyp	L/S	Eigenschaft
DefaultButton	boolean	L+S	Standardschaltfläche

Navigationsleiste

Englische Bezeichnung: *NavigationBar*

Weitere Eigenschaften und Methoden, die mit der Navigation zusammenhängen – z.B. Filter und das Ändern des Datensatzzeigers –, werden über das Formular geregelt.

Name	Datentyp	L/S	Eigenschaft
IconSize	short	L+S	Symbolgröße.
ShowPosition	boolean	L+S	Positionierung anzeigen und eingeben.
ShowNavigation	boolean	L+S	Navigation ermöglichen.
ShowRecordActions	boolean	L+S	Datensatzaktionen ermöglichen.
ShowFilterSort	boolean	L+S	Filter und Sortierung ermöglichen.

Methoden bei Formularen und Kontrollfeldern

Die Datentypen der Parameter werden durch Kürzel angedeutet:

- c Nummer der Spalte des gewünschten Feldes in der Datenmenge – ab 1 gezählt
- n numerischer Wert – je nach Situation als ganze Zahl oder als Dezimalzahl
- s Zeichenkette (String); die maximale Länge ergibt sich aus der Tabellendefinition
- b *boolean* (Wahrheitswert) – *true* (wahr) oder *false* (falsch)
- d Datumswert

In einer Datenmenge navigieren

Diese Methoden gelten sowohl für ein Formular als auch für die Ergebnismenge einer Abfrage.

Mit «Cursor» ist in den Beschreibungen der Datensatzzeiger gemeint.

Name	Datentyp	Beschreibung
Prüfungen für die Position des Cursors		
isBeforeFirst	boolean	Der Cursor steht vor der ersten Zeile, wenn der Cursor nach dem Einlesen noch nicht gesetzt wurde.
isFirst	boolean	Gibt an, ob der Cursor auf der ersten Zeile steht.
isLast	boolean	Gibt an, ob der Cursor auf der letzten Zeile steht.
isAfterLast	boolean	Der Cursor steht hinter der letzten Zeile, wenn er von der letzten Zeile aus mit *next* weiter gesetzt wurde.
getRow	long	Nummer der aktuellen Zeile
Setzen des Cursors Beim Datentyp boolean steht das Ergebnis «true» dafür, dass das Navigieren erfolgreich war.		
beforeFirst	–	Wechselt vor die erste Zeile.
first	boolean	Wechselt zur ersten Zeile.
previous	boolean	Geht um eine Zeile zurück.
next	boolean	Geht um eine Zeile vorwärts.

Name	Datentyp	Beschreibung
last	boolean	Wechselt zur letzten Zeile.
afterLast	–	Wechselt hinter die letzte Zeile.
absolute(n)	boolean	Geht zu der Zeile mit der angegebenen Nummer.
relative(n)	boolean	Geht um eine bestimmte Anzahl von Zeilen weiter: bei positivem Wert von n vorwärts, andernfalls zurück.
Maßnahmen zum Status der aktuellen Zeile		
refreshRow	–	Liest die ursprünglichen Werte der aktuellen Zeile neu ein.
rowInserted	boolean	Gibt an, ob es sich um eine neue Zeile handelt.
rowUpdated	boolean	Gibt an, ob die aktuelle Zeile geändert wurde.
rowDeleted	boolean	Gibt an, ob die aktuelle Zeile gelöscht wurde.

Datenzeilen bearbeiten

Die Methoden zum Lesen stehen bei jedem Formular und bei einer Ergebnismenge zur Verfügung. Die Methoden zum Ändern und Speichern gibt es nur bei einer Datenmenge, die geändert werden kann (in der Regel also nur bei Tabellen, nicht bei Abfragen).

Name	Datentyp	Beschreibung
Maßnahmen für die ganze Zeile		
insertRow	–	Speichert eine neue Zeile.
updateRow	–	Bestätigt Änderungen der aktuellen Zeile.
deleteRow	–	Löscht die aktuelle Zeile.
cancelRowUpdates	–	Macht Änderungen der aktuellen Zeile rückgängig.
moveToInsertRow	–	Wechselt den Cursor in die Zeile für einen neuen Datensatz.
moveToCurrentRow	–	Kehrt nach der Eingabe eines neuen Datensatzes zurück zur vorherigen Zeile.
Werte lesen		
getString(c)	string	Liefert den Inhalt der Spalte als Zeichenkette.
getBoolean(c)	boolean	Liefert den Inhalt der Spalte als Wahrheitswert.
getByte(c)	byte	Liefert den Inhalt der Spalte als einzelnes Byte.
getShort(c)	short	Liefert den Inhalt der Spalte als ganze Zahl.
getInt(c)	integer	
getLong(c)	long	
getFloat(c)	float	Liefert den Inhalt der Spalte als Dezimalzahl von einfacher Genauigkeit.
getDouble(c)	double	Liefert den Inhalt der Spalte als Dezimalzahl von doppelter Genauigkeit. – Wegen der automatischen Konvertierung durch Basic ist dies auch für decimal- und currency-Werte geeignet.
getBytes(c)	array of bytes	Liefert den Inhalt der Spalte als Folge einzelner Bytes.

Name	*Datentyp*	*Beschreibung*
getDate(c)	Date	Liefert den Inhalt der Spalte als Datumswert.
getTime(c)	Time	Liefert den Inhalt der Spalte als Zeitwert.
getTimestamp(c)	DateTime	Liefert den Inhalt der Spalte als Zeitstempel (Datum und Zeit).
In Basic selbst werden Datums- und Zeitwerte einheitlich mit dem Datentyp DATE verarbeitet. Für den Zugriff auf die Datenmenge gibt es verschiedene Datentypen: com.sun.star.util.Date für ein Datum, com.sun.star.util.Time für eine Zeit, com.sun.star.util.DateTime für einen Zeitstempel.		
wasNull	boolean	Gibt an, ob der Wert der zuletzt gelesenen Spalte NULL war.
Werte speichern		
updateNull(c)	–	Setzt den Inhalt der Spalte c auf NULL.
updateBoolean(c,b)	–	Setzt den Inhalt der Spalte c auf den Wahrheitswert b.
updateByte(c,x)	–	Speichert in Spalte c das angegebene Byte x.
updateShort(c,n)	–	Speichert in Spalte c die angegebene ganze Zahl n.
updateInt(c,n)	–	
updateLong(c,n)	–	
updateFloat(c,n)	–	Speichert in Spalte c die angegebene Dezimalzahl n.
updateDouble(c,n)	–	
updateString(c,s)	–	Speichert in Spalte die angegebene Zeichenkette s.
updateBytes(c,x)	–	Speichert in Spalte das angegebene Byte-Array x.
updateDate(c,d)	–	Speichert in Spalte das angegebene Datum d.
updateTime(c,d)	–	Speichert in Spalte den angegebenen Zeitwert d.
updateTimestamp(c,d)	–	Speichert in Spalte den angegebenen Zeitstempel d.

Einzelne Werte bearbeiten

Mit diesen Methoden wird über **`BoundField`** aus einem Kontrollfeld der Inhalt der betreffenden Spalte gelesen oder geändert. Diese Methoden entsprechen fast vollständig denen im vorigen Abschnitt; die Angabe der Spalte entfällt.

Name	*Datentyp*	*Beschreibung*
Werte lesen		
getString	string	Liefert den Inhalt der Spalte als Zeichenkette.
getBoolean	boolean	Liefert den Inhalt der Spalte als Wahrheitswert.
getByte	byte	Liefert den Inhalt der Spalte als einzelnes Byte.
getShort	short	Liefert den Inhalt der Spalte als ganze Zahl.
getInt	integer	
getLong	long	
getFloat	float	Liefert den Inhalt der Spalte als Dezimalzahl von einfacher Genauigkeit.
getDouble	double	Liefert den Inhalt der Spalte als Dezimalzahl von doppelter Genauigkeit. – Wegen der automatischen Konvertierung durch Basic ist dies auch für decimal- und currency-Werte geeignet.
getBytes	array of bytes	Liefert den Inhalt der Spalte als Folge einzelner Bytes.
getDate	Date	Liefert den Inhalt der Spalte als Datumswert.
getTime	Time	Liefert den Inhalt der Spalte als Zeitwert.
getTimestamp	DateTime	Liefert den Inhalt der Spalte als Zeitstempel (Datum und Zeit).
In Basic selbst werden Datums- und Zeitwerte einheitlich mit dem Datentyp DATE verarbeitet. Für den Zugriff auf die Datenmenge gibt es verschiedene Datentypen: com.sun.star.util.Date für ein Datum, com.sun.star.util.Time für eine Zeit, com.sun.star.util.DateTime für einen Zeitstempel.		
wasNull	boolean	Gibt an, ob der Wert der zuletzt gelesenen Spalte NULL war.
Werte speichern		
updateNull	–	Setzt den Inhalt der Spalte auf NULL.
updateBoolean(b)	–	Setzt den Inhalt der Spalte auf den Wahrheitswert b.
updateByte(x)	–	Speichert in der Spalte das angegebene Byte x.
updateShort(n)	–	Speichert in der Spalte die angegebene ganze Zahl n.
updateInt(n)	–	
updateLong(n)	–	
updateFloat(n)	–	Speichert in der Spalte die angegebene Dezimalzahl n.
updateDouble(n)	–	
updateString(s)	–	Speichert in der Spalte die angegebene Zeichenkette s.
updateBytes(x)	–	Speichert in der Spalte das angegebene Byte-Array x.
updateDate(d)	–	Speichert in der Spalte das angegebene Datum d.
updateTime(d)	–	Speichert in der Spalte den angegebenen Zeitwert d.
updateTimestamp(d)	–	Speichert in der Spalte den angegebenen Zeitstempel d.

Parameter für vorbereitete SQL-Befehle

Die Methoden, mit denen die Werte einem vorbereiteten SQL-Befehl – siehe «*Vorbereitete SQL-Befehle mit Parametern*» – übergeben werden, sind ähnlich denen der vorigen Abschnitte. Der erste Parameter – mit i bezeichnet – nennt seine Nummer (Position) innerhalb des SQL-Befehls.

Name	***Datentyp***	***Beschreibung***
setNull(i, n)	–	Setzt den Inhalt der Spalte auf NULL n bezeichnet den SQL-Datentyp gemäß *API-Referenz*.
setBoolean(i, b)	–	Fügt den angegebenen Wahrheitswert b in den SQL-Befehl ein.
setByte(i, x)	–	Fügt das angegebene Byte x in den SQL-Befehl ein.
setShort(i, n)	–	Fügt die angegebene ganze Zahl n in den SQL-Befehl ein.
setInt(i, n)		
setLong(i, n)		
setFloat(i, n)	–	Fügt die angegebene Dezimalzahl n in den SQL-Befehl ein.
setDouble(i, n)		
setString(i, s)	–	Fügt die angegebene Zeichenkette s in den SQL-Befehl ein.
setBytes(i, x)	–	Fügt das angegebene Byte-Array x in den SQL-Befehl ein.
setDate(i, d)	–	Fügt das angegebene Datum d in den SQL-Befehl ein.
setTime(i, d)	–	Fügt den angegebenen Zeitwert d in den SQL-Befehl ein.
setTimestamp(i, d)	–	Fügt den angegebenen Zeitstempel d in den SQL-Befehl ein.
clearParameters	–	Entfernt die bisherigen Werte aller Parameter eines SQL-Befehls.

Bedienbarkeit verbessern

Als erste Kategorie werden verschiedene Möglichkeiten vorgestellt, die zur Verbesserung der Bedienbarkeit von Base-Formularen dienen. Sofern nicht anders erwähnt, sind diese Makros Bestandteil der **Beispieldatenbank** «Medien_mit_Makros.odb».

Automatisches Aktualisieren von Formularen

Oft wird in einem Formular etwas geändert und in einem zweiten, auf der gleichen Seite liegenden Formular, soll die Änderung anschließend erscheinen. Hier hilft bereits ein kleiner Codeschnipsel, um das betreffende Anzeigeformular zu aktualisieren.

```
SUB Aktualisieren
```

Zuerst wird einmal das Makro benannt. Die Standardbezeichnung für ein Makro ist **SUB**. Dies kann groß oder klein geschrieben sein, Mit **SUB** wird eine Prozedur ablaufen gelassen, die nach außen keinen Wert weitergibt. Weiter unten wird im Gegensatz dazu einmal eine Funktion beschrieben, die im Unterschied dazu Rückgabewerte erzeugt.

Das Makro hat jetzt den Namen «*Aktualisieren*». Um sicher zu gehen, dass keine Variablen von außen eingeschleust werden gehen viele Programmierer so weit, dass sie Basic über **Option**

Explicit gleich zu Beginn mitteilen: Erzeuge nicht automatisch irgendwelche Variablen, sondern nutze nur die, die ich auch vorher definiert habe.

Deshalb werden jetzt standardgemäß erst einmal die Variablen deklariert. Bei allen hier deklarierten Variablen handelt es sich um Objekte (nicht z.B. Zahlen oder Texte), so dass der Zusatz **AS OBJECT** hinter der Deklaration steht. Um später noch zu erkennen, welchen Typ eine Variable hat, ist vor die Variablenbezeichnung ein «o» gesetzt worden. Prinzipiell ist aber die Variablenbezeichnung nahezu völlig frei wählbar.

```
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
```

Das Formular liegt in dem momentan aktiven Dokument. Der Behälter, in dem alle Formulare aufbewahrt werden, wird als **Drawpage** bezeichnet. Im Formularnavigator ist dies sozusagen der oberste Begriff, an den dann sämtliche Formulare angehängt werden.

Das Formular, auf das zugegriffen werden soll, ist hier mit den Namen "Anzeige" versehen. Dies ist der Name, der auch im Formularnavigator sichtbar ist. So hat z.B. das erste Formular standardmäßig erst einmal den Namen "MainForm".

```
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.forms.getByName("Anzeige")
```

Nachdem das Formular jetzt ansprechbar gemacht wurde und der Punkt, an dem es angesprochen wurde, in der Variablen **oForm** gespeichert wurde, wird es jetzt mit dem Befehl **reload()** neu geladen.

```
    oForm.reload()
END SUB
```

Die Prozedur hat mit **SUB** begonnen. Sie wird mit **END SUB** beendet.

Dieses Makro kann jetzt z.B. ausgelöst werden, wenn die Abspeicherung in einem anderen Formular erfolgt. Wird z.B. in einem Kassenformular an einer Stelle die Anzahl der Gegenstände und (über Barcodescanner) die Nummer eingegeben, so kann in einem anderen Formular im gleichen geöffneten Fenster hierdurch der Kassenstand, die Bezeichnung der Ware usw. nach dem Abspeichern sichtbar gemacht werden.

Filtern von Datensätzen

Der Filter selbst funktioniert ja schon ganz ordentlich in einer weiter oben beschriebenen Variante im Kapitel «Datenfilterung». Die untenstehende Variante ersetzt den Abspeicherungsbutton und liest die Listenfelder neu ein, so dass ein gewählter Filter aus einem Listenfeld die Auswahl in dem anderen Listenfeld einschränken kann.[2]

```
SUB Filter
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm1 AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oFeldList1 AS OBJECT
    DIM oFeldList2 AS OBJECT
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
```

Zuerst werden die Variablen definiert und auf das Gesamtformular zugegriffen. Das Gesamtformular besteht aus den Formularen "Filter" und "Anzeige". Die Listenfelder befinden sich in dem Formular "Filter" und sind mit dem Namen "Liste_1" und "Liste_2" versehen.

```
    oForm1 = oDrawpage.forms.getByName("Filter")
    oForm2 = oDrawpage.forms.getByName("Anzeige")
```

2 Siehe zu diesem Abschnitt auch die Datenbank «Beispiel_Suchen_und_Filtern.odb», die diesem Handbuch beiliegt.

```
    oFeldList1 = oForm1.getByName("Liste_1")
    oFeldList2 = oForm1.getByName("Liste_2")
```

Zuerst wird der Inhalt der Listenfelder an das darunterliegende Formular mit **`commit()`** weitergegeben. Die Weitergabe ist notwendig, da ansonsten die Änderung eines Listenfeldes bei der Speicherung nicht berücksichtigt wird. Genau genommen müsste der **`commit()`** nur auf dem Listenfeld ausgeführt werden, das gerade betätigt wurde. Danach wird der Datensatz mit **`updateRow()`** abgespeichert. Es existiert ja in unserer Filtertabelle prinzipiell nur ein Datensatz, und der wird zu Beginn einmal geschrieben. Dieser Datensatz wird also laufend durch ein Update-Kommado überschrieben.

```
    oFeldList1.commit()
    oFeldList2.commit()
    oForm1.updateRow()
```

Die Listenfelder sollen einander beeinflussen. Wird in einem Listenfeld z.B. eingegrenzt, dass an Medien nur CDs angezeigt werden sollen, so muss das andere Listenfeld bei den Autoren nicht noch sämtliche Buchautoren auflisten. Eine Auswahl im 2. Listenfeld hätte dann allzu häufig ein leeres Filterergebnis zur Folge. Daher müssen die Listenfelder jetzt neu eingelesen werden. Genau genommen müsste der **`refresh()`** nur auf dem Listenfeld ausgeführt werden, das gerade nicht betätigt wurde.

Anschließend wird das Formular2, das den gefilterten Inhalt anzeigen soll, neu geladen.

```
    oFeldList1.refresh()
    oFeldList2.refresh()
    oForm2.reload()
END SUB
```

Soll mit diesem Verfahren ein Listenfeld von der Anzeige her beeinflusst werden, so kann das Listenfeld mit Hilfe verschiedener Abfragen bestückt werden.

Die einfachste Variante ist, dass sich die Listenfelder mit ihrem Inhalt aus dem Filterergebnis versorgen. Dann bestimmt der eine Filter, aus welchen Datenbestand anschließend weiter gefiltert werden kann.

```
SELECT
   "Feld_1" || ' - ' || "Anzahl" AS "Anzeige",
   "Feld_1"
FROM
   ( SELECT COUNT( "ID" ) AS "Anzahl", "Feld_1" FROM "Suchtabelle"
   GROUP BY "Feld_1" )
ORDER BY "Feld_1"
```

Es wird der Feldinhalt und die Trefferzahl angezeigt. Um die Trefferzahl zu errechnen, wird eine Unterabfrage gestellt. Dies ist notwendig, da sonst nur die Trefferzahl ohne weitere Information aus dem Feld in der Listbox angezeigt würde.

Das Makro erzeugt durch dieses Vorgehen ganz schnell Listboxen, die nur noch mit einem Wert gefüllt sind. Steht eine Listbox nicht auf NULL, so wird sie schließlich bei der Filterung bereits berücksichtigt. Nach Betätigung der 2. Listbox stehen also bei beiden Listboxen nur noch die leeren Felder und jeweils 1 angezeigter Wert zur Verfügung. Dies mag für eine eingrenzende Suche erst einmal praktisch erscheinen. Was aber, wenn z.B. in einer Bibliothek die Zuordnung zur Systematik klar war, aber nicht eindeutig, ob es sich um ein Buch, eine CD oder eine DVD handelt? Wurde einmal die Systematik angewählt und dann die 2. Listbox auf CD gestellt so muss, um auch die Bücher zu sehen, die 2. Listbox erst einmal wieder auf NULL gestellt werden, um dann auch die Bücher anwählen zu können. Praktischer wäre, wenn die 2. Listbox direkt die verschiedenen Medienarten anzeigen würde, die zu der Systematik zur Verfügung stehen – natürlich mit den entsprechenden Trefferquoten.

Um dies zu erreichen, wurde die folgende Abfrage konstruiert, die jetzt nicht mehr direkt aus dem Filterergebnis gespeist wird. Die Zahlen für die Treffer müssen anders ermittelt werden.

```
SELECT
```

```
   COALESCE( "Feld_1" || ' - ' || "Anzahl", 'leer - ' || "Anzahl" ) AS
      "Anzeige",
   "Feld_1"
FROM
   ( SELECT COUNT( "ID" ) AS "Anzahl", "Feld_1" FROM "Tabelle"
      WHERE "ID" IN
      ( SELECT "Tabelle"."ID" FROM "Filter", "Tabelle"
         WHERE "Tabelle"."Feld_2" = COALESCE( "Filter"."Filter_2",
         "Tabelle"."Feld_2" ) )
      GROUP BY "Feld_1"
   )
ORDER BY "Feld_1"
```

Diese doch sehr verschachtelte Abfrage kann auch unterteilt werden. In der Praxis bietet es sich häufig an, die Unterabfrage in einer Tabellenansicht (**'VIEW'**) zu erstellen. Das Listenfeld bekommt seinen Inhalt dann über eine Abfrage, die sich auf diesen **'VIEW'** bezieht.

Die Abfrage im Einzelnen:

Die Abfrage stellt 2 Spalten dar. Die erste Spalte enthält die Ansicht, die die Person sieht, die das Formular vor sich hat. In der Ansicht werden die Inhalte des Feldes und, mit einem Bindestrich abgesetzt, die Treffer zu diesem Feldinhalt gezeigt. Die zweite Spalte gibt ihren Inhalt an die zugrundeliegende Tabelle des Formulars weiter. Hier steht nur der Inhalt des Feldes. Die Listenfelder beziehen ihre Inhalte dabei aus der Abfrage, die als Filterergebnis im Formular dargestellt wird. Nur diese Felder stehen schließlich zur weiteren Filterung zur Verfügung.

Als Tabelle, aus der diese Informationen gezogen werden, liegt eine Abfrage vor. In dieser Abfrage werden die Primärschlüsselfelder gezählt (**SELECT COUNT("ID") AS "Anzahl"**). Dies geschieht gruppiert nach der Bezeichnung, die in dem Feld steht (**GROUP BY "Feld_1"**). Als zweite Spalte stellt diese Abfrage das Feld selbst als Begriff zur Verfügung. Diese Abfrage wiederum basiert auf einer weiteren Unterabfrage:

```
SELECT "Tabelle"."ID"
FROM "Filter", "Tabelle"
WHERE "Tabelle"."Feld_2" = COALESCE( "Filter"."Filter_2",
   "Tabelle"."Feld_2" )
```

Diese Unterabfrage bezieht sich jetzt auf das andere zu filternde Feld. Prinzipiell muss das andere zu filternde Feld auch zu den Primärschlüsselnummern passen. Sollten noch mehrere weitere Filter existieren, so ist diese Unterabfrage zu erweitern:

```
SELECT "Tabelle"."ID"
FROM "Filter", "Tabelle"
WHERE "Tabelle"."Feld_2" = COALESCE( "Filter"."Filter_2",
      "Tabelle"."Feld_2" )
   AND
   "Tabelle"."Feld_3" = COALESCE( "Filter"."Filter_3",
      "Tabelle"."Feld_3" )
```

Alle weiteren zu filternden Felder beeinflussen, was letztlich in dem Listenfeld des ersten Feldes, "Feld_1", angezeigt wird.

Zum Schluss wird die gesamte Abfrage nur noch nach dem zugrundeliegenden Feld sortiert.

Wie letztlich die Abfrage aussieht, die dem anzuzeigenden Formular zugrunde liegt, ist im Kapitel «Datenfilterung» nachzulesen.

Mit dem folgenden Makro kann über das Listenfeld gesteuert werden, welches Listenfeld abgespeichert werden muss und welches neu eingelesen werden muss.

Die Variablen für das Array werden in den Eigenschaften des Listenfeldes unter Zusatzinformationen abgelegt. Die erste Variable enthält dort immer den Namen des Listenfeldes selbst, die weiteren Variablen die Namen aller anderen Listenfelder, getrennt durch Kommata.

```
SUB Filter_Zusatzinfo(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm1 AS OBJECT
    DIM oForm2 AS OBJECT
    DIM stTag AS String
    stTag = oEvent.Source.Model.Tag
```

Ein Array (Ansammlung von Daten, die hier über Zahlenverbindungen abgerufen werden können) wird gegründet und mit den Feldnamen der Listenfelder gefüllt. Der erste Name ist der Name des Listenfelds, das mit der Aktion (Event) verbunden ist.

```
    aList() = Split(stTag, ",")
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm1 = oDrawpage.forms.getByName("Filter")
    oForm2 = oDrawpage.forms.getByName("Anzeige")
```

Das Array wird von seiner Untergrenze (**`LBound()`**) bis zu seiner Obergrenze (**`UBound()`**) in einer Schleife durchlaufen. Alle Werte, die in den Zusatzinformationen durch Komma getrennt erschienen, werden jetzt nacheinander weitergegeben.

```
    FOR i = LBound(aList()) TO UBound(aList())
        IF i = 0 THEN
```

Das auslösende Listenfeld muss abgespeichert werden. Es hat die Variable **`aList(0)`**. Zuerst wird die Information des Listenfeldes auf die zugrundeliegende Tabelle übertragen, dann wird der Datensatz gespeichert.

```
            oForm1.getByName(aList(i)).commit()
            oForm1.updateRow()
        ELSE
```

Die anderen Listenfelder müssen neu eingelesen werden, da sie ja in Abhängigkeit vom ersten Listenfeld jetzt andere Werte abbilden.

```
            oForm1.getByName(aList(i)).refresh()
        END IF
    NEXT
    oForm2.reload()
END SUB
```

Die Abfragen für dieses besser nutzbare Makro sind natürlich die gleichen wie in diesem Abschnitt zuvor bereits vorgestellt.

Daten über den Formularfilter filtern

Alternativ zu dieser Vorgehensweise ist es auch möglich, die Filterfunktion des Formulars direkt zu bearbeiten.

```
SUB FilterSetzen
    DIM oDoc AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld  AS OBJECT
    DIM stFilter As String
    oForm = thisComponent.Drawpage.Forms.getByName("MainForm")
    oFeld = oForm.getByName("Filter")
    stFilter = oFeld.Text
    oForm.filter = " UPPER(""Name"") LIKE '%'||'" + UCase(stFilter) + "'||'%'"
    oForm.ApplyFilter = TRUE
    oForm.reload()
End Sub
```

Das Feld wird im Formular aufgesucht, der Inhalt ausgelesen. Der Filter wird entsprechend gesetzt. Die Filterung wird angeschaltet und das Formular neu geladen.

```
SUB FilterEntfernen
    DIM oForm AS OBJECT
    oForm = thisComponent.Drawpage.Forms.getByName("MainForm")
    oForm.ApplyFilter = False
    oForm.reload()
END SUB
```

Die Beendigung des Filters kann natürlich auch über die Navigationsleiste erfolgen. In diesem Fall wird einfach ein weiteres Makro genutzt.

Über diese Filterfunktion kann ein Formular auch direkt mit einem Filter z. B. für nur einen Datensatz gestartet werden. Aus dem startenden Formular wird ein Wert (z.B. ein Primärschlüssel für den aktuellen Datensatz) ausgelesen und an das Zielformular als Filterwert weiter gegeben.

Durch Datensätze mit der Bildlaufleiste scrollen

Die Bildlaufleiste lässt sich nur über Makros nutzen. Das folgende Beispiel[3] zeigt auf, wie mit so einer Bildlaufleiste durch Datensätze gescrollt werden kann. Die Bildlaufleiste kann dann statt der Navigationsleiste zur Navigation durch die Datensätze genutzt werden.

```
GLOBAL loPos AS LONG
```

Die Position des aktuellen Datensatzes wird als globale Variable gespeichert, damit sie aus allen Prozeduren gelesen und in allen Prozeduren geändert werden kann.

```
SUB MaxRow(oEvent AS OBJECT)
    'Auslösen durch das Formular "Beim Laden" und "Nach der Datensatzaktion"
    DIM oForm AS OBJECT
    DIM oScrollField AS OBJECT
    DIM loMax AS LONG
    oForm = oEvent.Source
    oScrollField = oForm.getByName("Bildlaufleiste")
    loPos = oForm.getRow
    oForm.last
    loMax = oForm.getRow
    oForm.absolute(loPos)
    oScrollField.ScrollValueMax = loMax
    oScrollField.ScrollValue = loPos
END SUB
```

Die Gesamtzahl der Datensätze kann nicht über die Funktion **`RowCount`** des Formulars ermittelt werden, da diese Funktion nur die Datensätze zählt, die bereits in den Cache geladen wurden. Deswegen wird hier zuerst die aktuelle Zeilennummer des Formulars ausgelesen, dann ans Ende der einzulesenden Daten gesprungen und die dortige Zeilennummer als Maximalwert ermittelt. Anschließend muss wieder auf die ursprüngliche Zeile mit **`oForm.absolute()`** zurückgesprungen werden.

Der Bildlaufleiste wird der ermittelte maximale Wert als **`ScrollValueMax`** und die aktuelle Position als **`loPos`** mitgeteilt. Geschieht das letzte nicht, so stimmt die Position innerhalb der Bildlaufleiste nicht unbedingt mit dem aktuellen Datensatz überein.

```
SUB Navigation(oEvent AS OBJECT)
    'Auslösen durch das Formular "Nach dem Datensatzwechsel"
    'Synchronisiert die Scrollstellung mit der Position des Datensatzes
    DIM oForm AS OBJECT
    DIM oScrollField AS OBJECT
    oForm = oEvent.Source
    oScrollField = oForm.getByName("Bildlaufleiste")
    loPos = oForm.getRow
    IF loPos = 0 THEN
        'Bei einem neuen Datensatz wird über getRow '0' ermittelt.
        'In dem Datensatzanzeiger soll stattdessen die maximale Zahl an Zeilen
        ''RowCount' um '1' erhöht werden
        loPos = oForm.RowCount + 1
```

3 Die Beispieldatenbank «Beispiel_Datensatz_scrollbar.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt.

```
    END IF
    oScrollField.ScrollValue = loPos
END SUB
```

Wird durch die Datensätze navigiert, so muss die Anzeige der Bildlaufleiste und die Anzeige in der Navigationsleiste immer übereinstimmen. Deshalb wird nach dem Datensatzwechsel immer die aktuelle Zeilennummer ermittelt. Für die aktuelle Zeilennummer wird '0' ausgegeben, wenn der Cursor zur Neuaufnahme eines Datensatzes über die letzte Zeile hinaus geht. In diesem Fall soll aber die Bildlaufleiste nicht auf die Startposition zurückspringen sondern wie die Navigationsleiste um '1' oberhalb des bisherigen maximalen Wertes positioniert werden.

```
SUB FormScroll(oEvent AS OBJECT)
    'Auslösen durch die Bildlaufleiste "Beim Justieren"
    DIM oForm AS OBJECT
    DIM oScrollAction AS OBJECT
    oScrollAction = oEvent.Source
    oForm = oScrollAction.Model.Parent
    loPos = oScrollAction.getValue()
    oForm.absolute(loPos)
END SUB
```

In dieser Prozedur wird aus der Bildlaufleiste ein neuer Wert für die Zeile des Formulars ermittelt. Über **`getValue()`** wird der Wert aus der Bildlaufleiste ausgelesen und dem Formular über **`oForm.absolute(loPos)`** zugewiesen.

Daten aus Textfeldern auf SQL-Tauglichkeit vorbereiten

Beim Speichern von Daten über einen SQL-Befehl können vor allem Hochkommata **`(')`** Probleme bereiten, wie sie z.B. in Namensbezeichnungen wie O'Connor vorkommen können. Dies liegt daran, dass Texteingaben in Daten in **`''`** eingeschlossen sind. Hier muss eine Funktion eingreifen und die Daten entsprechend vorbereiten.

```
FUNCTION String_to_SQL(st AS STRING)
    IF InStr(st,"'") THEN
        st = Join(Split(st,"'"),"''")
    END IF
    String_to_SQL = st
END FUNCTION
```

Es handelt sich hier um eine Funktion. Eine Funktion nimmt einen Wert auf und liefert anschließend auch einen Gegenwert zurück.

Der übergebende Text wird zuerst einmal daraufhin untersucht, ob er ein Hochkomma enthält. Ist dies der Fall, so wird der Text an der Stelle aufgetrennt - der Trenner dafür ist das Hochkomma – und anschließend stattdessen mit zwei Hochkommata wieder zusammengefügt. Der SQL-Code wird so maskiert.

Die Funktion übergibt ihr Ergebnis durch den folgenden Aufruf:

```
stTextneu = String_to_SQL(stTextalt)
```

Es wird also einfach nur die Variable stTextalt überarbeitet und der entsprechende Wert wieder in der Variablen stTextneu gespeichert. Dabei müssen die Variablen gar nicht unterschiedlichen heißen. Der Aufruf geht praktischer direkt mit:

```
stText = String_to_SQL(stText)
```

Diese Funktion wird in den nachfolgenden Makros immer wieder benötigt, damit Hochkommata auch über SQL abgespeichert werden können.

Beliebige SQL-Kommandos speichern und bei Bedarf ausführen

Über das Abfragemodul können nur Abfragen gestartet werden. SQL-Code, der auch verändernd auf Daten wirkt, ist dort nicht ausführbar, sofern nicht der Datenbanktreiber das, wie bei dem direkten Treiber für MySQL, zulässt. Der über **Extras → SQL** zur Verfügung stehende Editor lässt zwar

die Ausführung von Code zu, bietet aber keine Speichermöglichkeit. Laufend wiederkehrende Befehle müssen so umständlich irgendwo separat abgespeichert werden und können dann in den Editor über die Zwischenablage kopiert werden. Das folgende Makro schafft hier Abhilfe.[4]

Der für eine Operation wie **UPDATE**, **DELETE** oder **INSERT** gedachte Code wird in einer Tabelle abgelegt, deren erstes Feld den Primärschlüssel **"ID"** und deren zweites Feld den SQL-Code enthält. Der jeweils aktuelle Datensatz wird in einem Formular ausgewählt und dann über einen Button ausgeführt.

```
SUB ChangeData(oEvent AS OBJECT)
    DIM oConnection AS OBJECT
    DIM oForm AS OBJECT
    DIM stSql AS STRING
    DIM oSql_Statement AS OBJECT
    DIM inValue AS INTEGER
    oForm = oEvent.Source.Model.Parent
    oConnection = oForm.activeConnection()
    stSQL = oForm.getString(2)
    inValue = MsgBox("Soll der SQL-Code" & CHR(13) & stSQL & CHR(13) & "ausgeführt
        werden?", 20, "SQL-Code ausführen")
    IF inValue = 6 THEN
        oSQL_Statement = oConnection.createStatement()
        oSQL_Statement.executeUpdate(stSql)
    END IF
END SUB
```

Nach der Definition der Variablen wird das aktuelle Formular über das auslösende Ereignis des Buttons ermittelt. Die Verbindung zur Datenbank wird aus dem Formular ausgelesen. Als zweites Feld steht in der Tabelle der gewünschte SQL-Code.

Zur Sicherheit wird dieser SQL-Code zuerst in einer Messagebox noch einmal dargestellt. Wird hier nicht mit Ja bestätigt, so wird der Code nicht ausgeführt. Diese Bestätigung wird über den Rückgabewert der Messagebox ermittelt (**6** entspricht Ja). Anschließend wird zuerst die Verbindung für das Statement hergestellt und dann das Statement ausgeführt.

Werte in einem Formular vorausberechnen

Werte, die über die Datenbankfunktionen berechnet werden können, werden in der Datenbank nicht extra gespeichert. Die Berechnung erfolgt allerdings nicht während der Eingabe im Formular, sondern erst nachdem der Datensatz abgespeichert ist. Solange das Formular aus einem Tabellenkontrollfeld besteht, mag das nicht so viel ausmachen. Schließlich kann direkt nach der Eingabe ein berechneter Wert ausgelesen werden. Bei Formularen mit einzelnen Feldern bleibt der vorherige Datensatz aber nicht unbedingt sichtbar. Hier bietet es sich an, die Werte, die sonst in der Datenbank berechnet werden, direkt in entsprechenden Feldern anzuzeigen.[5]

Die folgenden drei Makros zeigen, wie so etwas vom Prinzip her ablaufen kann. Beide Makros sind mit dem Verlassen bestimmter Felder gekoppelt. Dabei ist auch berücksichtigt, dass hinterher eventuell Werte in einem bereits bestehenden Feld geändert werden.

```
SUB Berechnung_ohne_MWSt(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeld2 AS OBJECT
    oFeld = oEvent.Source.Model
    oForm = oFeld.Parent
    oFeld2 = oForm.getByName("Preis_ohne_MWSt")
    oFeld2.BoundField.UpdateDouble(oFeld.getCurrentValue / 1.19)
    IF NOT IsEmpty(oForm.getByName("Anzahl").getCurrentValue()) THEN
        Berechnung_gesamt2(oForm.getByName("Anzahl"))
    END IF
```

4 Die Beispieldatenbank «Beispiel_InsertUpdateDelete_SQL.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt.

5 Siehe hierzu die Beispieldatenbank «Beispiel_Direktberechnung_im Formular.odb»

```
END SUB
```

Ist in einem Feld «Preis» ein Wert eingegeben, so wird beim Verlassen des Feldes das Makro ausgelöst. Im gleichen Formular wie das Feld «Preis» liegt das Feld «Preis_ohne_MWSt». Für dieses Feld wird mit **`BoundField.UpdateDouble`** der berechnete Preis ohne Mehrwertsteuer festgelegt. Das Datenfeld dazu entstammt einer Abfrage, bei der vom Prinzip her die gleiche Berechnung, allerdings bei bereits gespeicherten Daten, durchgeführt wird. Auf diese Art und Weise wird der berechnete Wert sowohl während der Eingabe als auch später während der Navigation durch die Datensätze sichtbar, ohne abgespeichert zu werden.

Ist bereits im Feld «Anzahl» ein Wert enthalten, so wird eine Folgerechnung auch für die damit verbundenen Felder durchgeführt.

```
SUB Berechnung_gesamt(oEvent AS OBJECT)
    oFeld = oEvent.Source.Model
    Berechnung_gesamt2(oFeld)
END SUB
```

Diese kurze Prozedur dient nur dazu, die Auslösung der Folgeprozedur vom Verlassen des Formularfeldes «Anzahl» weiter zu geben. Die Angabe könnte genauso gut mit Hilfe der Bestimmung des Feldes über die Drawpage in der Folgeprozedur integriert werden.

```
SUB Berechnung_gesamt2(oFeld AS OBJECT)
    DIM oForm AS OBJECT
    DIM oFeld2 AS OBJECT
    DIM oFeld3 AS OBJECT
    DIM oFeld4 AS OBJECT
    oForm = oFeld.Parent
    oFeld2 = oForm.getByName("Preis")
    oFeld3 = oForm.getByName("Preis_gesamt_mit_MWSt")
    oFeld4 = oForm.getByName("MWSt_gesamt")
    oFeld3.BoundField.UpdateDouble(oFeld.getCurrentValue * oFeld2.getCurrentValue)
    oFeld4.BoundField.UpdateDouble(oFeld.getCurrentValue * oFeld2.getCurrentValue -
        oFeld.getCurrentValue * oFeld2.getCurrentValue / 1.19)
END SUB
```

Diese Prozedur ist lediglich eine Prozedur, bei der mehrere Felder berücksichtigt werden sollen. Die Prozedur wird aus einem Feld «Anzahl» gestartet, das die Anzahl bestimmter gekaufter Waren vorgeben soll. Mit Hilfe dieses Feldes und des Feldes «Preis» wird jetzt der «Preis_gesamt_mit_MWSt» und die «MWSt_gesamt» berechnet und in die entsprechenden Felder übertragen.

Nachteil in den Prozeduren und auch bei Abfragen: Der Steuersatz wird hier fest einprogrammiert. Besser wäre eine entsprechende Angabe dazu in Verbindung mit dem Preis, da ja Steuersätze unterschiedlich sein können und auch nicht immer konstant sind. In dem Fall müsste eben der Mehrwertsteuersatz aus einem Feld des Formulars ausgelesen werden.

Die aktuelle Office-Version ermitteln

Mit der Version 4.1 sind Änderungen bei Listenfeldern und Datumswerten vorgenommen worden, die es erforderlich machen, vor der Ausführung eines Makros für diesen Bereich zu erkunden, welche Office-Version denn nun verwendet wird. Dazu dient der folgende Code:

```
FUNCTION OfficeVersion()
    DIM aSettings, aConfigProvider
    DIM aParams2(0) AS NEW com.sun.star.beans.PropertyValue
    DIM sProvider$, sAccess$
    sProvider = "com.sun.star.configuration.ConfigurationProvider"
    sAccess = "com.sun.star.configuration.ConfigurationAccess"
    aConfigProvider = createUnoService(sProvider)
    aParams2(0).Name = "nodepath"
    aParams2(0).Value = "/org.openoffice.Setup/Product"
    aSettings = aConfigProvider.createInstanceWithArguments(sAccess, aParams2())
    OfficeVersion() = array(aSettings.ooName,aSettings.ooSetupVersionAboutBox)
END FUNCTION
```

Diese Funktion gibt eine Array wieder, das als ersten Wert z.B. "LibreOffice" und als zweiten Wert die detaillierte Version, z.B. "4.1.5.2" ausgibt.

Wert von Listenfeldern ermitteln

Mit LibreOffice 4.1 wird der Wert, den Listenfelder an die Datenbank weitergeben, über «CurrentValue» ermittelt. In Vorversionen, auch OpenOffice oder AOO, ist dies nicht der Fall. Die folgende Funktion soll dem Rechnung tragen. Die ermittelte LO-Version muss daraufhin untersucht werden, ob sie nach der Version 4.0 entstanden ist.

```
FUNCTION ID_Ermittlung(oFeld AS OBJECT) AS INTEGER
    a() = OfficeVersion()
    IF a(0) = "LibreOffice" AND ((LEFT(a(1),1) = 4 AND RIGHT(LEFT(a(1),3),1) > 0) OR
        LEFT(a(1),1) > 4) THEN
        stInhalt = oFeld.currentValue
    ELSE
```

Vor LO 4.1 wird der Wert, der weiter gegeben wird, aus der Werteliste des Listenfeldes ausgelesen. Der sichtbar ausgewählte Datensatz ist SelectedItems(0). '0', weil auch mehrere Werte in einem Listenfeld ausgewählt werden könnten.

```
        stInhalt = oFeld.ValueItemList(oFeld.SelectedItems(0))
    END IF
    IF IsEmpty(stInhalt) THEN
```

Mit -1 wird ein Zahlenwert weiter gegeben, der nicht als AutoWert verwendet wird, also in vielen Tabellen nicht als Fremdschlüssel existiert.

```
        ID_Ermittlung = -1
    ELSE
        ID_Ermittlung = Cint(stInhalt)
```

Umwandlung von Text in Integer

```
    END IF
END FUNCTION
```

Die Funktion gibt den Wert als Integer wieder. Meist werden für Primärschlüssel ja automatisch hoch zählende Integer-Werte verwendet. Für eine Verwendung von Fremdschlüsseln, die diesem Kriterium nicht entsprechen, muss die Ausgabe der Variablen entsprechend angepasst werden.

Der angezeigte Wert eines Listenfeldes lässt sich weiterhin über die Ansicht des Feldes ermitteln:

```
SUB Listenfeldanzeige
    DIM oDoc AS OBJECT
    DIM oForm AS OBJECT
    DIM oListbox AS OBJECT
    DIM oController AS OBJECT
    DIM oView AS OBJECT
    oDoc = thisComponent
    oForm = oDoc.Drawpage.Forms(0)
    oListbox = oForm.getByName("Listenfeld")
    oController = oDoc.getCurrentController()
    oView = oController.getControl(oListbox)
    print "Angezeigter Inhalt: " & oView.SelectedItem
END SUB
```

Es wird über den Controller auf die Ansicht des Formulars zugegriffen. Damit wird ermittelt, was auf der sichtbaren Oberfläche tatsächlich erscheint. Der ausgewählte Wert ist der **`SelectedItem`**.

Listenfelder durch Eingabe von Anfangsbuchstaben einschränken

Manchmal kann es vorkommen, dass der Inhalt für Listenfelder unübersichtlich groß wird. Damit eine Suche schneller zum Erfolg führt, wäre es sinnvoll, hier den Inhalt des Listenfeldes nach Eingabe eines oder mehrerer Buchstaben einzugrenzen. Das Listenfeld selbst wird erst einmal mit einem SQL-Befehl versehen, der nur als Platzhalter dient. Hier könnte z.B. stehen:

```
SELECT "Name", "ID" FROM "Tabelle" ORDER BY "Name" LIMIT 5 (HSQLDB)
SELECT "Name", "ID" FROM "Tabelle" ORDER BY "Name" ROWS 5 (FIREBIRD)
```

So wird beim Öffnen des Formulars vermieden, dass Base erst einmal die umfangreiche Liste einlesen muss.

Das folgende Makro ist dafür an **Eigenschaften: Listenfeld → Ereignisse → Taste losgelassen** gekoppelt.

```
GLOBAL stListStart AS STRING
GLOBAL lZeit AS LONG
```

Zuerst werden globale Variablen erstellt. Diese Variablen sind notwendig, damit nicht nur nach einem Buchstaben, sondern nach dem Betätigen weiterer Tasten schließlich auch nach einer Buchstabenkombination gesucht werden kann.

In der globalen Variablen **`stListStart`** werden die Buchstaben in der eingegebenen Reihenfolge gespeichert.

Die globale Variable **`lZeit`** wird mit der aktuellen Zeit in Sekunden versorgt. Bei einer längeren Pause zwischen den Tastatureingaben soll die Variable **`stListStart`** wieder zurückgesetzt werden können. Deswegen wird jeweils der Zeitunterschied zur vorhergehenden Eingabe abgefragt.

```
SUB ListFilter(oEvent AS OBJECT)
    oFeld = oEvent.Source.Model
    IF oEvent.KeyCode < 538 OR oEvent.KeyCode = 1283 OR oEvent.KeyCode = 1284 THEN
```

Das Makro wird durch einen Tastendruck ausgelöst. Eine Taste hat innerhalb der API einen bestimmten Zahlencode, der unter *com>sun>star>awt>Key* nachgeschlagen werden kann. Sonderzeichen wie das «ä», «ö» und «ü» haben den **`KeyCode`** 0, alle anderen Schriftzeichen und Zahlen haben einen **`KeyCode`** kleiner als 538. Den **`KeyCode`** 1283 belegt die Backspace-Taste. Wird dieser Code mit ausgelesen, so können auch Korrekturen durchgeführt werden. Mit dem **`KeyCode`** 1284 wird auch die Leertaste in die möglichen Zeichen aufgenommen.

Die Abfrage des **`KeyCode`** ist hier wichtig, da auch der Schritt mit der Tabulatortaste auf das Auswahlfeld natürlich das Makro auslöst. Der **`KeyCode`** für die Tabulatortaste liegt allerdings bei 1282, so dass der weitere Code der Prozedur hier nicht ausgeführt wird.

```
        DIM stSql(0) AS STRING
```

Der SQL-Code für das Listenfeld wird in einem Array gespeichert. Das Array hat im Falle des SQL-Codes aber nur ein Datenfeld. Deshalb ist das Array direkt auf **`stSql(0)`** begrenzt.

Entsprechend muss auch beim Auslesen des SQL-Codes aus dem Listenfeld darauf geachtet werden, dass der SQL-Code nicht direkt als Text zugänglich ist. Stattdessen ist der Code in einem Array als einziger Eintrag vorhanden: **`oFeld.ListSource(0)`**.

Der SQL-Code wird nach der Deklaration der Variablen für die weitere Verwendung aufgesplittet. Für das Feld, das gefiltert werden soll, wird nach dem ersten Komma der Code abgetrennt. Das Feld muss also an der ersten Position stehen. Anschließend wird der verbleibende Teil an dem ersten erscheinenden Anführungsstrich «"» aufgetrennt. Damit beginnt die Feldbezeichnung. Diese Aufteilungen erfolgen hier mit einfachen Arrays. Der Variablen **`stFeld`** wird schließlich wieder das doppelte Anführungszeichen am Beginn hinzugefügt. Außerdem wird über **`Rtrim`** vermieden, dass eine eventuell noch vorhandene Leertaste am Schluss des Ausdrucks bestehen bleibt.

```
        DIM stText AS STRING
        DIM stFeld AS STRING
        DIM stQuery AS STRING
        DIM ar0()
        DIM ar1()
        ar0() = Split(oFeld.ListSource(0),",", 2)
        ar1() = Split(ar0(0),"""", 2)
        stFeld = """" & Rtrim(ar1(1))
```

In dem SQL-Code wird eine Sortieranweisung erwartet. Allerdings kann die Anweisung in SQL in Großbuchstaben, Kleinbuchstaben oder beliebig gemischt erfolgen. Deshalb wird hier nicht mit

Split, sondern mit Hilfe der Funktion **inStr** nach der Zeichenkette «ORDER» gesucht. Der abschließende Parameter in dieser Funktion sagt mit der «1» aus, dass nicht nach Groß- und Kleinschreibung unterschieden werden soll. Alles, was links von der Zeichenkette «ORDER» steht, soll für die Konstruktion des neuen SQL-Codes weiter genutzt werden. Damit ist gewährleistet, dass auch Listenfelder bestückt werden können, die aus unterschiedlichen Tabellen oder über weitere Bedingungen im SQL-Code definiert worden sind.

```
stQuery = Left(oFeld.ListSource(0), inStr(1,oFeld.ListSource(0), "ORDER",1)-1)
IF inStr(stQuery, "LOWER") > 0 THEN
   stQuery = Left(stQuery, inStr(stQuery, "LOWER")-1)
ELSEIF inStr(1,stQuery, "WHERE",1) > 0 THEN
   stQuery = stQuery & " AND "
ELSE
   stQuery = stQuery & " WHERE "
END IF
```

Enthält die ermittelte Abfrage den Begriff «LOWER», so wird davon ausgegangen, dass die Abfrage bereits über die Prozedur **ListFilter** erstellt wurde. Deswegen wird die neu zu konstruierende Abfrage nur bis zur Position dieses Begriffes übernommen.

Ist dies nicht der Fall und es existiert in der Abfrage bereits der Begriff «WHERE» in beliebiger Schreibweise, so müssen weitere Bedingungen an die Abfrage mit **AND** angehängt werden.

Sind beide Bedingungen nicht erfüllt, so wird ein **WHERE** an den bestehenden Code angehängt.

```
IF lZeit > 0 AND Timer() - lZeit < 5 THEN
   stListStart = stListStart & oEvent.KeyChar
ELSE
   stListStart = oEvent.KeyChar
END IF
lZeit = Timer()
```

Ist bereits einmal eine Zeit in der globalen Variablen abgespeichert worden und beträgt die Distanz zu dieser Zeit zum Zeitpunkt der Eingabe weniger als 5 Sekunden, so wird der eingegebene Buchstabe an die vorher eingegebenen Buchstaben angehängt. Anderenfalls wird der eingegebene Buchstabe als einzige (neue) Eingabe verstanden. Das Listenfeld wird dann einfach neu nach dem entsprechenden Buchstaben gefiltert. Anschließend wird die aktuelle Zeit wieder in der globalen Variablen **lZeit** gespeichert.

```
        stText = LCase( stListStart & "%")
        stSql(0) = stQuery + "LOWER("+stFeld+") LIKE '"+stText+"' ORDER BY "+stFeld+""
        oFeld.ListSource = stSql
        oFeld.refresh
    END IF
END SUB
```

Der SQL-Code wird schließlich zusammengefügt. Die Kleinschreibweise des Feldinhaltes wird mit der Kleinschreibweise des eingegebenen Buchstabens verglichen. Der Code wird dem Listenfeld hinzugefügt und das Listenfeld aufgefrischt, so dass nur noch der gefilterte Inhalt nachgeschlagen werden kann.

Datumswert aus einem Formularwert in eine Datumsvariable umwandeln

```
FUNCTION Datumswert(oFeld AS OBJECT) AS DATE
   a() = OfficeVersion()
   IF a(0) = "LibreOffice" AND (LEFT(a(1),1) = 4 AND RIGHT(LEFT(a(1),3),1) > 0)
      OR LEFT(a(1),1) > 4 THEN
```

Hier werden alle Versionen ab 4.1 durch die oben vorgestellte Funktion «OfficeVersion()» abgefangen. Dazu wird die Version in ihre Bestandteile aufgesplittet. Die Hauptversion und die erste Unterversion werden abgefragt. Das funktioniert vorerst bis zur LO-Version 9 einwandfrei.

```
            DIM stMonat AS STRING
            DIM stTag AS STRING
            stMonat = Right(Str(0) & Str(oFeld.CurrentValue.Month),2)
            stTag =  Right(Str(0) & Str(oFeld.CurrentValue.Day),2)
            Datumswert = CDateFromIso(oFeld.CurrentValue.Year & stMonat & stTag)
        ELSE
            Datumswert = CDateFromIso(oFeld.CurrentValue)
        END IF
    END FUNCTION
```

Das Datum wird seit LO 4.1.2 als Array im Formularfeld gespeichert. Mit dem aktuellen Wert kann also nicht auf das Datum selbst zugegriffen werden. Entsprechend ist es neu aus den Werten für Tag, Monat und Jahr zusammen zu setzen, damit anschließend in Makros damit weiter gearbeitet werden kann.

Suchen von Datensätzen

Ohne Makro funktioniert das Suchen von Datensätzen auch. Hier ist aber die entsprechende Abfrage äußerst unübersichtlich zu erstellen. Da könnte eine Schleife mittels Makro Abhilfe schaffen.

Die folgende Variante liest die Felder einer Tabelle aus, gründet dann intern eine Abfrage und schreibt daraus schließlich eine Liste der Primärschlüsselnummern der durchsuchten Tabelle auf, auf die der Suchbegriff zutrifft. Für die folgende Beschreibung existiert eine Tabelle "Suchtmp", die aus einem per Autowert erstellten Primärschlüsselfeld "ID" und einem Feld "Nr." besteht, in das die aus der zu durchsuchenden Tabelle gefundenen Primärschlüssel eingetragen werden. Der Tabellenname wird dabei der Prozedur am Anfang als Variable mitgegeben.

Um ein entsprechendes Ergebnis zu bekommen, muss die Tabelle natürlich nicht die Fremdschlüssel, sondern entsprechende Feldinhalte in Textform enthalten. Dafür ist gegebenenfalls eine Tabellenansicht (*View*) zu erstellen, auf den das Makro auch zugreifen kann.[6]

```
SUB Suche(stTabelle AS STRING)
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM stSql AS STRING
    DIM oAbfrageergebnis AS OBJECT
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oForm2 AS OBJECT
    DIM oFeld AS OBJECT
    DIM stInhalt AS STRING
    DIM arInhalt() AS STRING
    DIM inI AS INTEGER
    DIM inK AS INTEGER
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("Suchform")
    oFeld = oForm.getByName("Suchtext")
    stInhalt = oFeld.getCurrentValue()
    stInhalt = LCase(stInhalt)
```

Der Inhalt des Suchtext-Feldes wird hier von vornherein in Kleinbuchstaben umgewandelt, damit die anschließende Suchfunktion nur die Kleinschreibweisen miteinander vergleicht.

```
    oDatenquelle = ThisComponent.Parent.DataSource
    oVerbindung = oDatenquelle.GetConnection("","")
    oSQL_Anweisung = oVerbindung.createStatement()
```

6 Siehe zu diesem Abschnitt auch die Datenbank «Beispiel_Suchen_und_Filtern.odb», die diesem Handbuch beiliegt.

Zuerst wird einmal geklärt, ob überhaupt ein Suchbegriff eingegeben wurde. Ist das Feld leer, so wird davon ausgegangen, dass keine Suche vorgenommen wird. Alle Datensätze sollen angezeigt werden; eine weitere Abfrage erübrigt sich.

Ist ein Suchbegriff eingegeben worden, so werden die Spaltennamen der zu durchsuchenden Tabelle ausgelesen, um auf die Felder mit einer Abfrage zugreifen zu können.

```
IF stInhalt <> "" THEN
   stInhalt = String_to_SQL(stInhalt)
   stSql = "SELECT ""COLUMN_NAME"" FROM ""INFORMATION_SCHEMA"".""SYSTEM_COLUMNS""
      WHERE ""TABLE_NAME"" = '" + stTabelle + "' ORDER BY ""ORDINAL_POSITION"""
   oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

Hinweis

Der SQL-Code müsste für FIREBIRD hier angepasst werden:

```
stSql = "SELECT RDB$FIELD_NAME FROM RDB$RELATION_FIELDS WHERE
   RDB$RELATION_NAME = '" + stTabelle + "' ORDER BY RDB$Field_POSITION"
```

Auf die Doppelung der doppelten Anführungszeichen kann hier verzichtet werden, da die Bezeichnungen sowieso keine Sonderzeichen enthalten und nur aus Großbuchstaben zusammengesetzt sind.

Leider ist der folgende SQL-Code dieses Makros für Firebird so nicht geeignet, da Firebird nicht in der Lage ist, aus selektierten Daten eine neue Tabelle zu erstellen. (~~FIREBIRD~~)

Hinweis

SQL-Formulierungen müssen in Makros wie normale Zeichenketten zuerst einmal in doppelten Anführungsstrichen gesetzt werden. Feldbezeichnungen und Tabellenbezeichnungen stehen innerhalb der SQL-Formulierungen in der Regel bereits in doppelten Anführungsstrichen. Damit letztlich ein Code entsteht, der auch diese Anführungsstriche weitergibt, müssen für Feldbezeichnungen und Tabellenbezeichnungen diese Anführungsstriche verdoppelt werden.

Aus `stSql = "SELECT ""Name"" FROM ""Tabelle"";"`
wird, wenn es mit dem Befehl `msgbox stSql` auf dem Bildschirm angezeigt wird,
`SELECT "Name" FROM "Tabelle";`

Der Zähler des Arrays, in das die Feldnamen geschrieben werden, wird zuerst auf 0 gesetzt. Dann wird begonnen die Abfrage auszulesen. Da die Größe des Arrays unbekannt ist, muss immer wieder nachjustiert werden. Deshalb beginnt die Schleife damit, über **`ReDim Preserve arInhalt(inI)`** die Größe des Arrays festzulegen und den vorherigen Inhalt dabei zu sichern. Anschließend werden die Felder ausgelesen und der Zähler des Arrays um 1 heraufgesetzt. Damit kann dann das Array neu dimensioniert werden und wieder ein weiterer Wert abgespeichert werden.

```
InI = 0
WHILE oAbfrageergebnis.next
   ReDim Preserve arInhalt(inI)
   arInhalt(inI) = oAbfrageergebnis.getString(1)
   inI = inI + 1
WEND
stSql = "DROP TABLE ""Suchtmp"" IF EXISTS"
oSQL_Anweisung.executeUpdate (stSql)
```

Jetzt wird die Abfrage in einer Schleife zusammengestellt, die anschließend an die zu Beginn angegebene Tabelle gestellt wird. Dabei werden alle Schreibweisen untersucht, da auch der Inhalt des Feldes in der Abfrage auf Kleinbuchstaben umgewandelt wird.

Die Abfrage wird direkt so gestellt, dass die Ergebniswerte in der Tabelle "Suchtmp" landen. Dabei wird davon ausgegangen, dass der Primärschlüssel an der ersten Position der Tabelle steht (**`arInhalt(0)`**).

```
stSql = "SELECT """+arInhalt(0)+"""INTO ""Suchtmp"" FROM """+stTabelle+"""
   WHERE "
```

```
        FOR inK = 0 TO (inI - 1)
            stSql = stSql+"LOWER("""+arInhalt(inK)+""") LIKE '%"+stInhalt+"%'"
            IF inK < (inI - 1) THEN
                stSql = stSql+" OR "
            END IF
        NEXT
        oSQL_Anweisung.executeQuery(stSql)
    ELSE
        stSql = "DELETE FROM ""Suchtmp"""
        oSQL_Anweisung.executeUpdate (stSql)
    END IF
```

Das Anzeigeformular muss neu geladen werden. Es hat als Datenquelle eine Abfrage, in diesem Beispiel "Suchabfrage"

```
    oForm2 = oDrawpage.forms.getByName("Anzeige")
    oForm2.reload()
End Sub
```

Damit wurde eine Tabelle erstellt, die nun in einer Abfrage ausgewertet werden soll. Die Abfrage ist dabei möglichst so zu fassen, dass sie anschließend noch editiert werden kann. Im Folgenden also ein Abfragecode:

```
SELECT * FROM "Suchtabelle"
WHERE "Nr." IN ( SELECT "Nr." FROM "Suchtmp" )
   OR "Nr." =
      CASE WHEN ( SELECT COUNT( "Nr." ) FROM "Suchtmp" ) > 0
      THEN '0' ELSE "Nr." END
```

Alle Elemente der **"Suchtabelle"** werden dargestellt. Auch der Primärschlüssel. Keine andere Tabelle taucht in der direkten Abfrage auf; somit ist auch kein Primärschlüssel einer anderen Tabelle nötig, damit das Abfrageergebnis weiterhin editiert werden kann.

Der Primärschlüssel ist in dieser Beispieltabelle unter dem Titel **"Nr."** abgespeichert. Durch das Makro wird genau dieses Feld ausgelesen. Es wird jetzt also zuerst nachgesehen, ob der Inhalt des Feldes **"Nr."** in der Tabelle **"Suchtmp"** vorkommt. Bei der Verknüpfung mit **'IN'** werden ohne weiteres auch mehrere Werte erwartet. Die Unterabfrage darf also auch mehrere Datensätze liefern.

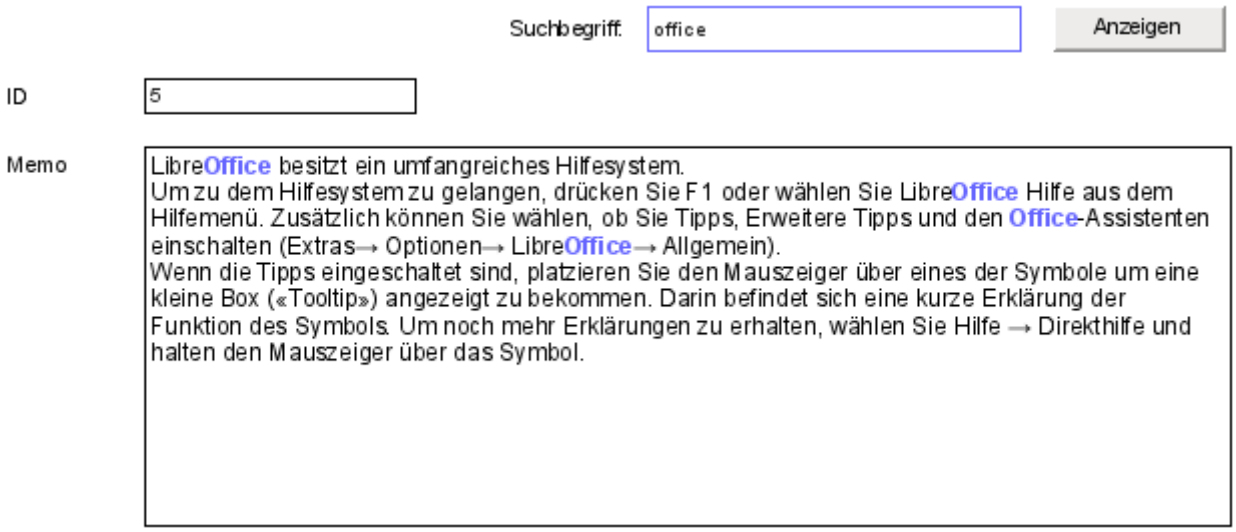
Bei größeren Datenmengen wird der Abgleich von Werten über die Verknüpfung IN aber zusehends langsamer. Es bietet sich also nicht an, für eine leere Eingabe in das Suchfeld einfach alle Primärschlüsselfelder der **"Suchtabelle"** in die Tabelle **"Suchtmp"** zu übertragen und dann auf die gleiche Art die Daten anzusehen. Stattdessen erfolgt bei einer leeren Eingabe eine Leerung der Tabelle **"Suchtmp"**, so dass gar keine Datensätze mehr vorhanden sind. Hierauf zielt der zweite Bedingungsteil:

```
OR "Nr." = CASE WHEN ( SELECT COUNT( "Nr." ) FROM "Suchtmp" ) > 0
           THEN '-1' ELSE "Nr." END
```

Wenn in der Tabelle "Suchtmp" ein Datensatz gefunden wird, so ist das Ergebnis der ersten Abfrage größer als 0. Für diesen Fall gilt: **"Nr." = '-1'** (hier steht am Besten ein Zahlenwert, der als Primärschlüssel nicht vorkommt, also z.B. **'-1'**). Ergibt die Abfrage genau 0 (Dies ist der Fall wenn keine Datensätze da sind), dann gilt **"Nr." = "Nr."**. Es wird also jeder Datensatz dargestellt, der eine **"Nr."** hat. Da **"Nr."** der Primärschlüssel ist, gilt dies also für alle Datensätze.

Suchen in Formularen und Ergebnisse farbig hervorheben

Bei größeren Inhalten eines Textfeldes ist oft unklar, an welcher Stelle denn nun die Suche den Treffer zu verzeichnen hat. Da wäre es doch gut, wenn das Formular den entsprechenden Treffer auch markieren könnte. So sollte das dann im Formular aussehen:



Um so ein Formular zum Laufen zu bekommen, bedarf es ein paar zusätzlicher Eingriffe in die Trickkiste.[7]

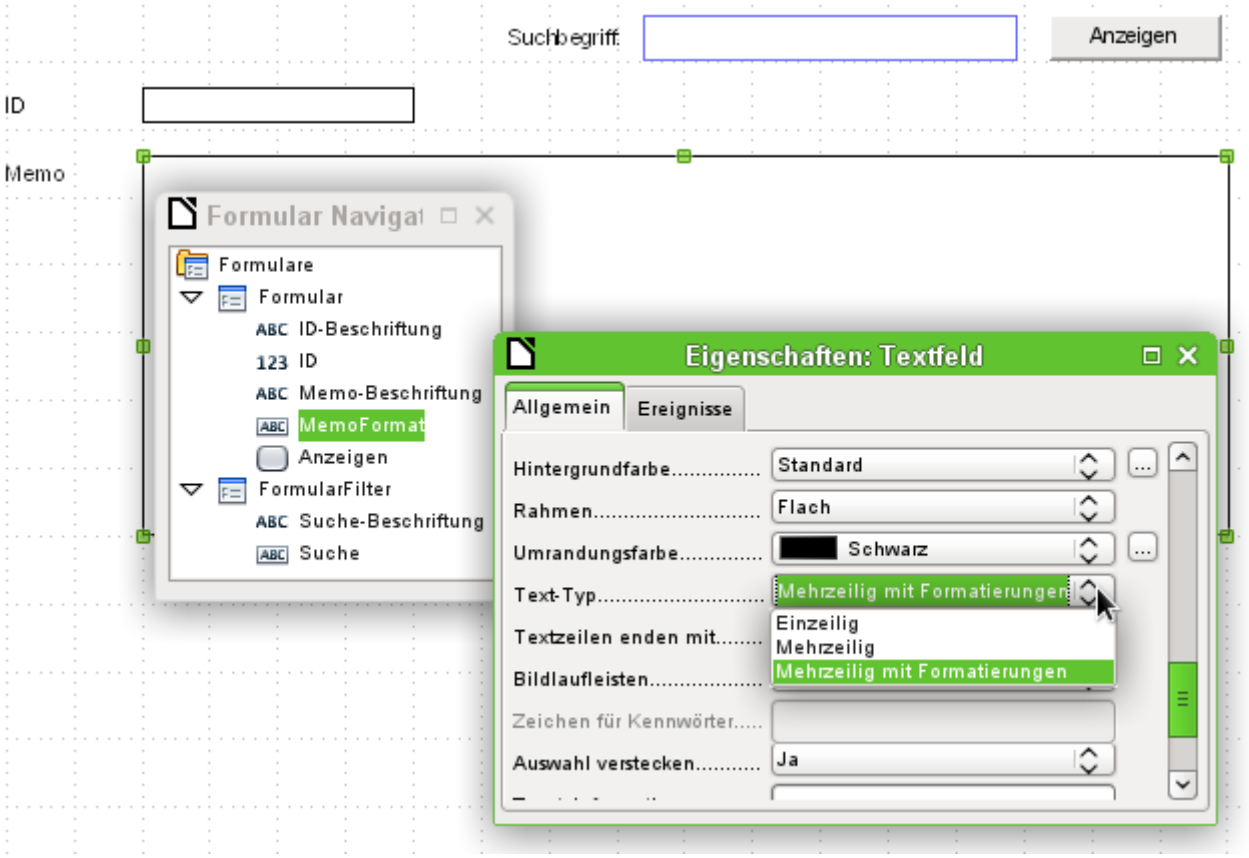
Die Funktionsweise so eines Suchfeldes wurde bereits bei den Abfragetechniken erklärt: Es wird eine Filtertabelle erstellt. Über ein Formular wird nur der aktuelle Wert des einzigen Datensatzes in dieser Tabelle neu geschrieben. Das Hauptformular wird über eine Abfrage mit dem entsprechenden Inhalt versorgt. Die Abfrage sieht im obigen Fall so aus:

```
SELECT "ID", "Memo"
FROM "Tabelle"
WHERE LOWER ( "Memo" ) LIKE '%' || LOWER (
   ( SELECT "Suchtext" FROM "Filter" WHERE "ID" = TRUE ) ) || '%'
```

Wird ein Suchtext eingetragen, so werden nur die Datensätze der Tabelle "Tabelle" angezeigt, bei denen der Text im Feld "Memo" vorkommt. Dabei ist Groß- und Kleinschreibung egal.

Wird kein Suchtext eingetragen, so werden alle Datensätze der Tabelle "Tabelle" angezeigt. Da der Primärschlüssel dieser Tabelle auch in der Abfrage enthalten ist, ist die Abfrage außerdem editierbar.

7 Siehe zu diesem Abschnitt auch die Datenbank «Beispiel_Autotext_Suchmarkierung_Rechtschreibung.odb», die dem Handbuch beiliegt.



In dem Formular ist neben dem Feld «ID» für den Primärschlüsseleintrag nur ein Feld «MemoFormat», das über **Eigenschaften → Allgemein → Text-Typ → Mehrzeilig mit Formatierungen** so eingestellt ist, dass es überhaupt Farben innerhalb von schwarzem Text darstellen kann. Die genaue Betrachtung der Eigenschaften des Textfeldes zeigt, dass der Reiter **Daten** fehlt. Daten lassen sich über ein Feld, das zusätzlich formatierbar ist, nicht eingeben. Das ist wohl dadurch begründet, dass die Datenbank selbst auch solche Formatierungen nicht speichert. Und trotzdem ist es durch den entsprechenden Makroeinsatz möglich, Text in dieses Feld hinein zu bekommen, ihn zu markieren und bei Änderungen auch wieder aus dem Feld hinaus in die Datenbank zu befördern.

Die Prozedur «InhaltUebertragen» dient dazu, den Inhalt aus dem Datenbankfeld "Memo" in das formatierbare Textfeld «MemoFormat» zu übertragen und so zu formatieren, dass bei einem entsprechenden Eintrag im Suchfeld der dazugehörige Begriff hervorgehoben wird.

Die Prozedur ist an das folgende Ereignis gebunden: **Formular → Ereignisse → Nach dem Datensatzwechsel**

```
Sub InhaltUebertragen(oEvent AS OBJECT)
    DIM inMemo AS INTEGER
    DIM oFeld AS OBJECT
    DIM stSuchtext AS STRING
    DIM oCursor AS OBJECT
    DIM inSuch AS INTEGER
    DIM inSuchAlt AS INTEGER
    DIM inLen AS INTEGER
    oForm = oEvent.Source
    inMemo = oForm.findColumn("Memo")
    oFeld = oForm.getByName("MemoFormat")
    oFeld.Text = oForm.getString(inMemo)
```

Zuerst werden die Variablen definiert. Anschließend wird über das Formular das Tabellenfeld "Memo" gesucht und aus diesem Feld über **`getString`**() der entsprechende Text des Feldes "Memo" der Tabelle "Tabelle" ausgelesen. Der entsprechende Feldinhalt wird in das Feld übertragen, das sich formatieren lässt, aber keine Verbindung zur Datenbank hat: «MemoFormat».

Bei Tests ist es zuerst vorgekommen, dass sich das Formular zwar öffnete, aber leider die Formularleiste am unteren Rand des Formulars nicht mehr aufgebaut wurde. Deswegen erfolgt hier ein sehr kurzer Wartebefehl von 5/1000 Sekunden. Danach wird aus dem parallel zum «Formular» liegenden «FormularFilter» der angezeigte Inhalt als Suchtext ausgelesen.

```
Wait 5
stSuchtext = oForm.Parent.getByName("FormularFilter").getByName("Suche").Text
```

Um Textteile formatieren zu können muss ein (nicht sichtbarer) **`TextCursor`** in dem Feld erstellt werden, das den Text enthält. Die Darstellung des Textes in der Standardversion hat eine serifenbetonte Schriftart in 12-Punkt-Größe, die in anderen Formularteilen nicht unbedingt vorkommt und über das Formularfeld auch nicht direkt abwählbar ist. In dieser Prozedur wird direkt zu Beginn der Text einmal auf die gewünschte Darstellungsart eingestellt. Erfolgt dies nicht schon zu Beginn, so wird wegen der unterschiedlichen Formatierungen der oberer Textrand in dem Feld erst einmal angeschnitten. Die erste Zeile war in Versuchen nur zu 2/3 lesbar.

Damit der Cursor (wieder nicht sichtbar) den Text markiert, wird er zuerst an den Anfang gesetzt und mit dem Zusatz **`true`** weiterbewegt zum Endpunkt, der ebenfalls den Zusatz **`true`** hat. Dann erfolgt die Zuweisung der notwendigen Eigenschaften wie Schriftgröße, Schriftstil, Schriftfarbe oder auch Schriftdicke. Anschließend wird der Cursor wieder zur Startposition gesetzt.

```
oCursor = oFeld.createTextCursor()
oCursor.gotoStart(true)
oCursor.gotoEnd(true)
oCursor.CharHeight = 10
oCursor.CharFontName = "Arial, Helvetica, Tahoma"
oCursor.CharColor = RGB(0,0,0)
oCursor.CharWeight = 100.000000   'com::sun::star::awt::FontWeight
oCursor.gotoStart(false)
```

Enthält das Feld Text und ist ein Eintrag zum Suchen vorhanden, so wird jetzt der Text nach dem Suchbegriff durchsucht. Die äußere Schleife fragt erst einmal nur nach der Bedingung, die nächste Schleife klärt noch einmal, ob der Suchtext denn tatsächlich in dem Text enthalten ist, der in «MemoFormat» steht. Diese Einstellung könnte auch unterlassen werden, da die Abfrage, auf der das Formular basiert, nur solchen Text anzeigt, auf den diese Bedingung zutrifft.

```
IF oFeld.Text <> "" AND stSuchtext <> "" THEN
   IF inStr(oFeld.Text, stSuchtext) THEN
      inSuch = 1
      inSuchAlt = 0
      inLen = Len(stSuchtext)
```

Der Text wird nach dem Suchtext durchsucht. Dies erfolgt in einer Schleife, die dann endet, wenn keine weitere Trefferposition mehr angezeigt wird. **`InStr`**() liefert dabei die Fundstelle des ersten Zeichens des Suchtextes, in der aufgezeigten Fassung unabhängig von Groß- und Kleinschreibung. Die Schleife wird dadurch gesteuert, dass der Suchbeginn **`inSuch`** bei jedem Schleifenende in der Summe um 1 erhöht wird (erste Schleifenzeile -1, letzte Schleifenzeile +2). Bei jedem Durchgang wird der Cursor mit **`oCursor.goRight(Position, false)`** zuerst ohne zu markieren an die Startstelle gesetzt, dann um die Länge des Suchtextes weiter mit der Markierungsaufforderung nach rechts gesetzt. Dann wird die gewünschte Formatierung (blau, etwas dicker) vorgenommen und der Cursor wieder für den nächsten Start an den Startpunkt der Markierung zurückgesetzt.

```
      DO WHILE inStr(inSuch, oFeld.Text, stSuchtext) > 0
         inSuch = inStr(inSuch, oFeld.Text, stSuchtext) - 1
         oCursor.goRight(inSuch-inSuchAlt,false)
         oCursor.goRight(inLen,true)
         oCursor.CharColor = RGB(102,102,255)
         oCursor.CharWeight = 110.000000
         oCursor.goLeft(inLen,false)
         inSuchAlt = inSuch
         inSuch = inSuch + 2
      LOOP
```

```
        END IF
    END IF
End Sub
```

Die Prozedur «InhaltSchreiben» dient dazu, den Inhalt aus dem formatierbaren Textfeld «Memo-Format» in die Datenbank zu übertragen. Dies erfolgt in dieser Fassung unabhängig davon, ob eine Änderung vorgenommen wurde.

Die Prozedur ist an das folgende Ereignis gebunden: **Formular → Ereignisse → Vor dem Datensatzwechsel**

```
Sub InhaltSchreiben(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    DIM inMemo AS INTEGER
    DIM loID AS LONG
    DIM oFeld AS OBJECT
    DIM stMemo AS STRING
    oForm = oEvent.Source
    IF InStr(oForm.ImplementationName, "ODatabaseForm") THEN
```

Das auslösende Ereignis ist doppelt belegt. Nur der Implementationsname, der auf **`ODatabaseForm`** endet, gibt den richtigen Zugriff auf den Datensatz.

```
        IF NOT oForm.isBeforeFirst() AND NOT oForm.isAfterLast() THEN
```

Beim Einlesen, auch beim Reload des Formulars, steht der Cursor vor dem ersten Datensatz. Würde jetzt ein Schreibversuch unternommen, dann erscheint die Meldung «ungüliger Cursorstatus».

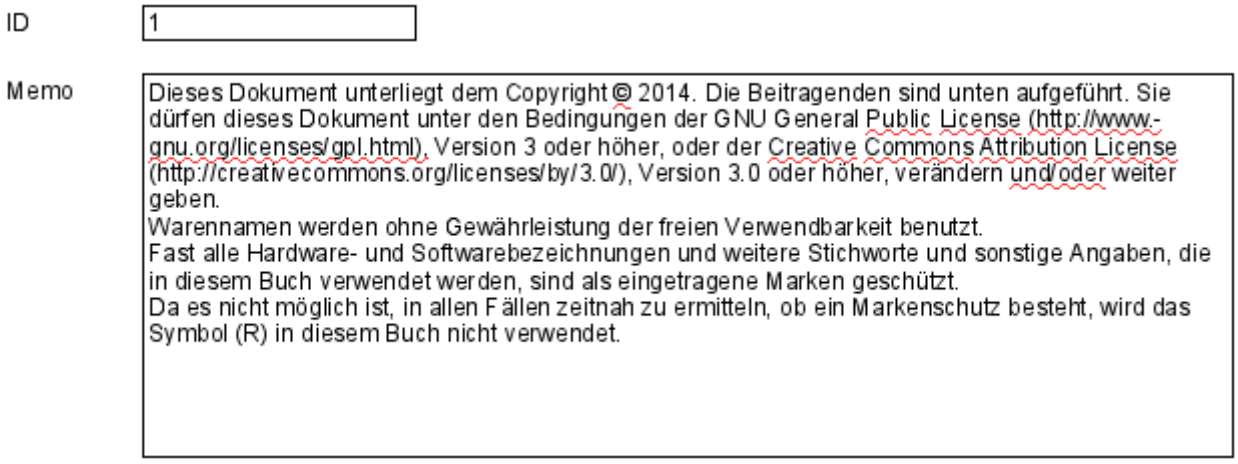
```
            inMemo = oForm.findColumn("Memo")
            loID = oForm.findColumn("ID")
            oFeld = oForm.getByName("MemoFormat")
            stMemo = oFeld.Text
            IF stMemo <> "" THEN
                oForm.updateString(inMemo,stMemo)
            END IF
            IF stMemo <> "" AND oForm.getString(loID) <> "" THEN
                oForm.UpdateRow()
            END IF
        END IF
    END IF
End Sub
```

Das Tabellenfeld "Memo" wird aus der Datenquelle des Formulars herausgesucht. Ebenso das Feld "ID". Befindet sich im Feld «MemoFormat» Text, so wird er mit **`oForm.updateString`**() in das Feld "Memo" der Datenquelle übertragen. Nur wenn bereits ein Eintrag im Feld "ID" existiert, also der Primärschlüssel belegt ist, erfolgt ein Update. Ansonsten wird ja sowieso ein neuer Datensatz über die Formularfunktionen eingefügt, da das Formular die Änderung entsprechend bemerkt und eine Abspeicherung selbständig vornimmt.

Rechtschreibkontrolle während der Eingabe

Auf **mehrzeilige Textfelder mit Formatierungen** greift auch dieses Makro zu. Entsprechend muss auch, wie bei dem vorherigen Kapitel, der Inhalt bei jedem Datensatzwechsel zuerst geschrieben und danach der Inhalt des neuen Datensatzes in das Formularfeld geladen werden. Die Prozeduren «InhaltUebertragen» und «InhaltSchreiben» unterscheiden sich höchstens in dem Punkt, dass die Suchfunktion ausgeklammert werden kann.[8]

8 Siehe auch hierzu: «Beispiel_Autotext_Suchmarkierung_Rechtschreibung.odb»



Die Rechtschreibkontrolle wird in dem obigen Formular dadurch ausgelöst, dass in dem Formularfeld entweder eine Leertaste oder ein Return betätigt wird. Sie läuft also nach der Beendigung eines Wortes jedes Mal ab und könnte gegebenenfalls auch noch mit dem Fokusverlust des Formularfeldes gekoppelt werden, damit auch das letzte Wort sicher überprüft wird.

Die Prozedur ist an das folgende Ereignis gebunden: **Formular → Ereignisse → Taste losgelassen**

```
SUB MarkierungFehlerDirekt(oEvent AS OBJECT)
   GlobalScope.BasicLibraries.LoadLibrary("Tools")
```

Es wird die Funktion **`RTrimStr`** zum Entfernen von Satzzeichen am Ende vor Worten benötigt. Sonst werden alle Worte, denen ein Komma, Punkt oder irgendein anderes Satzzeichen folgt, als falsch angesehen. Mit **`LTrimChar`** müssen außerdem Klammern zum Beginn des Wortes entfernt werden.

```
DIM aProp() AS NEW com.sun.star.beans.PropertyValue
DIM oLinuSvcMgr AS OBJECT
DIM oSpellChk AS OBJECT
DIM oFeld AS OBJECT
DIM arText()
DIM stWort AS STRING
DIM inlenWort AS INTEGER
DIM ink AS INTEGER
DIM i AS INTEGER
DIM oCursor AS OBJECT
DIM stText AS OBJECT
oLinguSvcMgr = createUnoService("com.sun.star.linguistic2.LinguServiceManager")
IF NOT IsNull(oLinguSvcMgr) THEN
   oSpellChk = oLinguSvcMgr.getSpellChecker()
END IF
```

Zuerst werden alle Variablen deklariert. Danach wird auf das Rechtschreibüberprüfungsmodul **`SpellChecker`** zugegriffen. Mit diesem Modul werden anschließend die einzelnen Worte auf ihre Richtigkeit hin überprüft.

```
oFeld = oEvent.Source.Model
ink = 0
IF oEvent.KeyCode = 1280 OR oEvent.KeyCode = 1284 THEN
```

Das Ereignis, das das Makro auslöst, ist ein Tastendruck. Zu dem Ereignis wird ein Code für jede Taste mitgeliefert, der **`KeyCode`**. Der **`KeyCode`** für die Returntaste ist 1280, der für die Leertaste ist 1284. Wie viele andere Informationen sind diese Informationen einfach durch das Tool «Xray» gewonnen worden. Wird also eine Leertaste oder die Returntaste betätigt, so wird die Rechtschreibung überprüft. Sie startet also zu jedem Wortende. Lediglich die Überprüfung für das letzte Wort ist so nicht automatisch möglich.

Bei jedem Durchlauf werden alle Worte des Textes überprüft. Die Überprüfung einzelner Worte könnte eventuell auch möglich sein, bedeutet aber erheblich mehr Aufwand.

Der Text wird also in Worte aufgesplittet. Trenner ist hier das Leerzeichen. Vorher müssen allerdings noch Trennungen an Zeilenumbrüchen erzeugt werden, die sonst später als ein Wort wahrgenommen werden.

```
        stText = Join(Split(oFeld.Text,CHR(10))," ")
        stText = Join(Split(stText,CHR(13))," ")
        arText = Split(RTrim(stText)," ")
        FOR i = LBound(arText) TO Ubound(arText)
            stWort = arText(i)
            inlenWort = len(stWort)
            stWort = Trim( RtrimStr( RtrimStr( RtrimStr( RtrimStr( RtrimStr(
                RtrimStr(stWort,","), "."),"?"),"!"),"."),")"))
            stWort = LTrimChar(stWort,"(")
```

Das einzelne Wort wird ausgelesen, seine ungekürzte Länge ist notwendig für die folgenden Bearbeitungsschritte. Nur so kann die Position des Wortes innerhalb des gesamten Textes bestimmt werden, die auch für die gezielte Markierung von Schreibfehlern gebraucht wird.

Mit **`Trim`** werden Leerzeichen entfernt, mit der Funktion **`RTrimStr`** Kommas und Satzzeichen am Ende des Textes, mit der Funktion **`LTrimChar`** Zeichen am Anfang des Textes.

```
            IF stWort <> "" THEN
                oCursor = oFeld.createTextCursor()
                oCursor.gotoStart(false)
                oCursor.goRight(ink,false)
                oCursor.goRight(inLenWort,true)
                If Not oSpellChk.isValid(stWort, "de", aProp()) Then
                    oCursor.CharUnderline = 9
                    oCursor.CharUnderlineHasColor = True
                    oCursor.CharUnderlineColor = RGB(255,51,51)
                ELSE
                    oCursor.CharUnderline = 0
                END IF
            END IF
            ink = ink + inLenWort + 1
        NEXT
    END IF
END SUB
```

Hat das Wort einen Inhalt, so wird zuerst einmal ein Textcursor erstellt. Der Textcursor wird ohne Markierung an den Start des Textes in dem Eingabefeld gesetzt. Dann geht es, immer noch ohne Markierung, um den Betrag nach rechts im Text vorwärts, der in der Variablen **`ink`** gespeichert ist. Diese Variable ist am Anfang 0, nach Durchlaufen der ersten Schleife dann so groß wie das vorhergehende Wort lang war +1 für das angehängte Leerzeichen. Dann wird der Cursor mit Markierung um die Länge des aktuellen Wortes weiter gesetzt. Erfolgt jetzt eine Änderung der Buchstabeneigenschaften, so betrifft diese nur den markierten Bereich.

Der **`Spellchecker`** startet. Als Variablen müssen das Wort und der Landescode übergeben werden. Ohne Landescode ist alles richtig. Das Array bleibt in der Regel leer.

Ist das Wort nicht in den Lexika eingetragen, so wird es mit einer roten Wellenlinie versehen. Die Wellenlinie entspricht hier der **`'9'`**. Ist das Wort eingetragen, so wird statt einer Wellenlinie keine Linie (**`'0'`**) gezeichnet. Dieser Schritt ist notwendig, weil sonst ein einmal als falsch erkanntes Wort bei einer Korrektur auch weiterhin mit der roten Wellenlinie gekennzeichnet würde. Eine rote Wellenlinie würde nie aufgehoben, da es keine entgegengesetzte Formatierung gibt.

Kombinationsfelder als Listenfelder mit Eingabemöglichkeit

Aus Kombinationsfeldern und Tabellenfeldern aus dem Formular kann direkt eine Tabelle mit einem Datensatz versehen und der entsprechende Primärschlüssel in eine andere Tabelle eingetragen werden.[9]

9 Die Beispieldatenbank «Beispiel_Combobox_Listfeld.odb» zum Einsatz von Kombinationsfeldern statt Listenfeldern ist den Beispieldatenbanken für dieses Handbuch beigefügt.

Das Modul «Comboboxen» macht aus den Formularfeldern zur Eingabe und Auswahl von Werten (Kombinationsfelder) Listenfelder mit Eingabemöglichkeiten. Dazu werden neben den Kombinationsfeldern im Formular die jeweils an die zugrundeliegende Tabelle zu übergebenden Schlüsselfeldwerte in den Tabellenspalten abgelegt, die dem Formular zugrunde liegen. Die Schlüssel aus den Tabellenspalten werden beim Start des Formulars ausgelesen und das Kombinationsfeld auf den entsprechenden Inhalt eingestellt. Wird der Inhalt des Kombinationsfeldes geändert, so wird er neu abgespeichert und der neue Primärschlüssel zum Abspeichern in der Haupttabelle in das entsprechende numerische Fremdschlüsselfeld übertragen.

Werden statt der Tabellen entsprechend konstruierte eingabefähige Abfragen erstellt, so kann der Text, den die Kombinationsfelder darstellen sollen, direkt aus den Abfragen ermittelt werden. Ein Makro ist dann für diesen Arbeitsschritt nicht notwendig.

Voraussetzung für die Funktionsweise des Makros ist, dass alle Primärschlüssel der Tabellen, die in den Kombinationsfeldern als Datenquellen auftauchen, mit einem automatisch hochzählenden Autowert versehen sind. Außerdem ist als Bezeichnung hier vorausgesetzt, dass die Primärschlüssel den Namen "ID" tragen.

Textanzeige im Kombinationsfeld

Diese Prozedur soll Text in den Kombinationsfeldern nach den Werten der (unsichtbaren) Fremdschlüssel-Felder aus dem Hauptformular einstellen. Dabei werden gegebenenfalls auch Listenfelder berücksichtigt, die sich auf 2 unterschiedliche Tabellen beziehen. Dies kann z.B. dann sein, wenn bei einer Ortsangabe die Postleitzahl vom Ort abgetrennt wurde. Dann wird die Postleitzahl aus einer Tabelle ausgelesen, in der auch ein Fremdschlüssel für den Ort liegt. Im Listenfeld werden Postleitzahl und Ort zusammen angezeigt.

```
SUB TextAnzeigen(oEvent AS OBJECT)
```

Dieses Makro sollte an das folgende Ereignis des Formulars gebunden werden: 'Nach dem Datensatzwechsel'

Das Makro wird direkt aus dem Formular angesprochen. Über das auslösende Ereignis werden die gesamten notwendigen Variablen für das Makro ermittelt.

Die Variablen werden deklariert. Einige Variablen sind in einem separaten Modul bereits global deklariert und werden hier nicht noch einmal erwähnt.

```
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeldList AS OBJECT
    DIM stAbfrage AS STRING
    DIM stFeldWert AS STRING
    DIM stFeldID AS STRING
    DIM inCom AS INTEGER
    oForm = oEvent.Source
```

Das Formular startet das Ereignis. Es ist die Quelle für das das Makro auslösende Ereignis.

In dem Formular befindet sich ein verstecktes Kontrollelement, aus dem hervorgeht, wie die verschiedenen Kombinationsfelder in diesem Formular heißen. Nacheinander werden dann in dem Makro die Kombinationsfelder abgearbeitet.

```
    aComboboxen() = Split(oForm.getByName("Comboboxen").Tag,",")
    FOR inCom = LBound(aComboboxen) TO Ubound(aComboboxen)
        ...
    NEXT inCom
```

Aus den Zusatzinformationen («Tag») des versteckten Kontrollelementes wird die Bezeichnung der Kombinationsfelder ermittelt. Sie sind dort durch Kommas voneinander getrennt aufgeschrieben. Die Namen der Felder werden in ein Array geschrieben und nacheinander in einer Schleife abgearbeitet. Die Schleife endet mit der Bezeichnung **`NEXT …`** .

Das Kombinationsfeld, das jetzt statt eines Listenfeldes existiert, wird anschließend als **`oFeldList`** bezeichnet. Der Fremdschlüssel wird über die Bezeichnung des Tabellenfeldes, die in

den Zusatzinformationen des Kombinationsfeldes steht, aus der Tabellenspalte des Formulars ermittelt.

```
oFeldList = oForm.getByName(Trim(aComboboxen(inCom)))
stFeldID = oForm.getString(oForm.findColumn(oFeldList.Tag))
oFeldList.Refresh()
```

Das Kombinationsfeld wird mit **`Refresh()`** neu eingelesen. Es kann ja sein, dass sich der Inhalt des Feldes durch Neueingaben geändert hat. Diese müssen schließlich verfügbar gemacht werden.

Die Abfrage, die zur Ermittlung des anzuzeigenden Inhaltes des Kombinationsfeldes notwendig ist, wird aus der Abfrage des Kombinationsfeldes und dem ermittelten Wert des Fremdschlüssels erstellt. Damit der SQL-Code brauchbar wird, wird zuerst eine eventuelle Sortierungsanweisung entfernt. Anschließend wird nachgesehen, ob bereits eine Beziehungsdefinition (beginnend mit **`WHERE`**) existiert. Da die **`InStr()`**-Funktion standardmäßig keinen Unterschied zwischen Groß- und Kleinschreibung macht, werden hier gleich alle Schreibweisen abgedeckt. Existiert eine Beziehungsdefinition, so enthält die Abfrage Felder aus zwei unterschiedlichen Tabellen. Es muss jetzt die Tabelle herausgesucht werden, aus der der Fremdschlüssel für die Verbindung zur Verfügung gestellt wird. Das Makro funktioniert hier nur mit Hilfe der Information, dass der Primärschlüssel einer jeden Tabelle "ID" heißt.

Existiert keine Beziehungsdefinition, so beruht die Abfrage nur auf einer Tabelle. Die Tabelleninformation kann entfallen, die Bedingung direkt mit dem Fremdschlüsselwert zusammen formuliert werden.

```
IF stFeldID <> "" THEN
    stAbfrage = oFeldList.ListSource
    IF InStr(stAbfrage,"order by") > 0 THEN
        stSql = Left(stAbfrage, InStr(stAbfrage,"order by")-1)
    ELSE
        stSql = stAbfrage
    END IF
    IF InStr(stSql,"where") THEN
        st = Right(stSql, Len(stSql)-InStr(stSql,"where")-4)
        IF InStr(Left(st, InStr(st,"=")),"."ID"") THEN
            a() = Split(Right(st, Len(st)-InStr(st,"=")-1),".")
        ELSE
            a() = Split(Left(st, InStr(st,"=")-1),".")
        END IF
            stSql = stSql + "AND "+a(0)+"."ID"" = "+stFeldID
    ELSE
        stSql = stSql + "WHERE "ID"" = "+stFeldID
    END IF
```

Jedes Feld und jeder Tabellenname muss bereits in der SQL-Eingabe mit doppelten Anführungsstrichen oben versehen werden. Da bereits Anführungsstriche einfacher Art in Basic als die Einführung zu Text interpretiert werden, sind diese bei der Weitergabe des Codes nicht mehr sichtbar. Erst bei einer Doppelung der Anführungsstriche wird ein Element mit einfachen Anführungsstrichen weitergegeben. **`""ID""`** bedeutet also, dass in der Abfrage auf das Feld **`"ID"`** (mit einfachen Anführungsstrichen für die SQL-Verbindung) zugegriffen wird.

Die in der Variablen **`stSql`** abgespeicherte Abfrage wird jetzt ausgeführt und das Ergebnis dieser Abfrage in der Variablen **`oAbfrageergebnis`** gespeichert.

```
oDatenquelle = ThisComponent.Parent.CurrentController
IF NOT (oDatenquelle.isConnected()) Then
    oDatenquelle.connect()
End IF
oVerbindung = oDatenquelle.ActiveConnection()
oSQL_Anweisung = oVerbindung.createStatement()
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

Das Abfrageergebnis wird über eine Schleife ausgelesen. Hier könnten, wie in einer Abfrage aus der GUI, mehrere Felder und Datensätze dargestellt werden. Von der Konstruktion der Abfrage her

wird aber nur ein Ergebnis erwartet. Dieses Ergebnis wird in der ersten Spalte **(1)** der Abfrage zu finden sein. Es ist der Datensatz, der den anzuzeigenden Inhalt des Kombinationsfeldes wiedergibt. Der Inhalt ist ein Textinhalt (**getString()**), deshalb hier **oAbfrageergebnis.getString(1)**.

```
            WHILE oAbfrageergebnis.next
                stFeldWert = oAbfrageergebnis.getString(1)
            WEND
```

Das Kombinationsfeld muss jetzt auf den aus der Abfrage sich ergebenden Textwert eingestellt werden.

```
                oFeldList.Text = stFeldWert
            ELSE
```

Falls überhaupt kein Wert in dem Feld für den Fremdschlüssel **oFeld** vorhanden ist, ist auch die ganze Abfrage nicht gelaufen. Das Kombinationsfeld wird jetzt auf eine leere Anzeige eingestellt.

```
                oFeldList.Text = ""
            END IF
        NEXT inCom
    END SUB
```

Diese Prozedur erledigt jetzt also den Kontakt von dem in der Datenquelle des Formulars abgelegten Fremdschlüssel zu dem Kombinationsfeld. Für die Anzeige der richtigen Werte im Kombinationsfeld würde das ausreichen. Eine Abspeicherung von neuen Werten hingegen benötigt eine weitere Prozedur.

Fremdschlüsselwert vom Kombinationsfeld zum numerischen Feld übertragen

Wird nun ein neuer Wert ausgewählt oder neu in das Kombinationsfeld eingegeben (nur wegen dieser Eigenschaft wurde ja das Makro konstruiert), so muss der entsprechende Primärschlüssel als Fremdschlüssel in die dem Formular zugrundeliegende Tabelle eingetragen werden.

```
SUB TextAuswahlWertSpeichern(oEvent AS OBJECT)
```

Dieses Makro sollte an das folgende Ereignis des Formulars gebunden werden: 'Vor der Datensatzaktion'.

Nach Deklaration der Variablen (hier nicht weiter aufgeführt) wird zuerst differenziert, bei welchem Ereignis genau das Makro überhaupt ablaufen soll. Vor der Datensatzaktion werden zwei Implementationen nacheinander aufgerufen. Für das Makro selbst ist es wichtig, das Formularobjekt zu erhalten. Das geht prinzipiell über beide Implementationen, aber eben auf unterschiedliche Weise. Es wird hier die Implementation mit dem Namen **"ODatabaseForm"** herausgefiltert.

```
    IF InStr(oEvent.Source.ImplementationName,"ODatabaseForm") THEN
        ...
    END IF
END SUB
```

In diese Schleife wird der gleiche Start wie bei der Prozedur **TextAnzeigen** eingebaut:

```
        oForm = oEvent.Source
        aComboboxen() = Split(oForm.getByName("Comboboxen").Tag,",")
        FOR inCom = LBound(aComboboxen) TO Ubound(aComboboxen)
            ...
        NEXT inCom
```

Das Feld **oFeldList** zeigt den Text an. Es kann in einem Tabellenkontrollfeld liegen. Dann kann nicht direkt vom Formular auf das Feld zugegriffen werden. In den Zusatzinformationen des versteckten Kontrollfeldes «Comboboxen» ist für diesen Fall der Pfad zum Kombinationsfeld über «Tabellenkontrollfeld>Kombinationsfeld» eingetragen. Durch Aufsplittung dieses Eintrages wird ermittelt, wie das Kombinationsfeld anzusprechen ist.

```
        a() = Split(Trim(aComboboxen(inCom)),">")
            IF Ubound(a) > 0 THEN
                oFeldList = oForm.getByName(a(0)).getByName(a(1))
            ELSE
```

```
        oFeldList = oForm.getByName(a(0))
    END IF
```

Anschließend wird die Abfrage aus dem Kombinationsfeld ausgelesen und in ihre Einzelteile zerlegt. Bei einfachen Kombinationsfeldern wären die notwendigen Informationen lediglich der Feldname und der Tabellenname:

```
SELECT "Feld" FROM "Tabelle"
```

Dies könnte gegebenenfalls noch durch eine Sortierung erweitert sein. Sobald zwei Felder in dem Kombinationsfeld zusammen dargestellt werden, muss aber bereits bei den Feldern zur Trennung entsprechend mehr Aufwand getrieben werden:

```
SELECT "Feld1"||' '||"Feld2" FROM "Tabelle"
```

Diese Abfrage fasst zwei Felder zusammen und setzt dazwischen eine Leertaste ein. Da der Trenner eine Leertaste ist, wird in dem Makro nach so einem Trenner gesucht und danach der Text in zwei Teile gesplittet. Das funktioniert natürlich nur dann einwandfrei, wenn "Feld1" nicht bereits Text enthalten soll, der eine Leertaste erlaubt. Sonst wird z.B. aus dem Vornamen «Anne Marie» und dem Nachnamen «Müller» durch das Makro der Vorname «Anne» und der Nachname «Marie Müller». Für solch einen Fall sollte ein passender Trenner eingesetzt werden, der dann auch vom Makro gefunden werden kann. Bei Namen ist dies z. B. ein Komma: «Nachname, Vorname».

Noch komplizierter wird es, wenn die beiden enthaltenen Felder aus zwei verschiedenen Tabellen stammen:

```
SELECT "Tabelle1"."Feld1"||' > '||"Tabelle2"."Feld2"
FROM "Tabelle1", "Tabelle2"
WHERE "Tabelle1"."ID" = "Tabelle2"."FremdID"
ORDER BY "Tabelle1"."Feld1"||' > '||"Tabelle2"."Feld2" ASC
```

Hier müssen die Felder voneinander getrennt, die Tabellenzuordnungen zu den Feldern erfasst und die Fremdschlüsselzuweisung ermittelt werden.

```
    stAbfrage = oFeldList.ListSource
    aFelder() = Split(stAbfrage, """")
    stInhalt = ""
    FOR i=LBound(aFelder)+1 TO UBound(aFelder)
```

Der Inhalt der Abfrage wird von Ballast befreit. Die Teile werden anschließend über eine nicht übliche Zeichenkombination zu einem Array wieder zusammengefügt.«FROM» trennt die sichtbare Feldanzeige von der Tabellenbezeichnung. «WHERE» trennt die Beziehungsdefinition von der Tabellenbezeichnung. Joins werden nicht unterstützt.

```
        IF Trim(UCASE(aFelder(i))) = "ORDER BY" THEN
            EXIT FOR
        ELSEIF Trim(UCASE(aFelder(i))) = "FROM" THEN
            stInhalt = stInhalt+" §§ "
        ELSEIF Trim(UCASE(aFelder(i))) = "WHERE" THEN
            stInhalt = stInhalt+" §§ "
        ELSE
        stInhalt = stInhalt+Trim(aFelder(i))
        END IF
    NEXT i
    aInhalt() = Split(stInhalt, " §§ ")
```

Die sichtbare Feldanzeige wird gegebenenfalls in Inhalte aus verschiedenen Feldern aufgeteilt:

```
    aErster() = Split(aInhalt(0),"||")
    IF UBound(aErster) > 0 THEN
        IF UBound(aInhalt) > 1 THEN
```

Der erste Teil enthält mindestens 2 Felder. Die Felder haben zu Beginn eine Tabellenbezeichnung. Der zweite Teil enthält zwei Tabellenbezeichnungen, die aber schon aus dem ersten Teil ermittelt werden können. Der dritte Teil enthält eine Beziehung über einen Fremdschlüssel mit «=» getrennt:

```
            aTest() = Split(aErster(0),".")
            NameTabelle1 = aTest(0)
```

```
            NameTabellenFeld1 = aTest(1)
            Erase aTest
            Feldtrenner = Join(Split(aErster(1),"'"),"")
            aTest() = Split(aErster(2),".")
            NameTabelle2 = aTest(0)
            NameTabellenFeld2 = aTest(1)
            Erase aTest
            aTest() = Split(aInhalt(2),"=")
            aTest1() = Split(aTest(0),".")
            IF aTest1(1) <> "ID" THEN
                NameTab12ID = aTest1(1)
                IF aTest1(0) = NameTabelle1 THEN
                    Position = 2
                ELSE
                    Position = 1
                END IF
            ELSE
                Erase aTest1
                aTest1() = Split(aTest(1),".")
                NameTab12ID = aTest1(1)
                IF aTest1(0) = NameTabelle1 THEN
                    Position = 2
                ELSE
                    Position = 1
                END IF
            END IF
        ELSE
```

Der erste Teil enthält zwei Feldbezeichnungen ohne Tabellenbezeichnung mit Trenner, der zweite Teil enthält die Tabellenbezeichnung. Ein dritter Teil ist nicht vorhanden:

```
            NameTabellenFeld1 = aErster(0)
            Feldtrenner = Join(Split(aErster(1),"'"),"")
            NameTabellenFeld2 = aErster(2)
            NameTabelle1 = aInhalt(1)
        END IF
    ELSE
```

Es existiert nur ein Feld, das aus einer Tabelle stammt:

```
        NameTabellenFeld1 = aErster(0)
        NameTabelle1 = aInhalt(1)
    END IF
```

Die maximale Zeichenlänge, die eine Eingabe haben darf, wird im Folgenden mit der Funktion **`Spaltengroesse`** ermittelt. Das Kombinationsfeld kann hier mit seiner Beschränkung nicht sicher weiterhelfen, da ja ermöglicht werden soll, gleichzeitig 2 Felder in einem Kombinationsfeld einzutragen.

```
    LaengeFeld1 = Spaltengroesse(NameTabelle1,NameTabellenFeld1)
    IF NameTabellenFeld2 <> "" THEN
        IF NameTabelle2 <> "" THEN
            LaengeFeld2 = Spaltengroesse(NameTabelle2,NameTabellenFeld2)
        ELSE
            LaengeFeld2 = Spaltengroesse(NameTabelle1,NameTabellenFeld2)
        END IF
    ELSE
        LaengeFeld2 = 0
    END IF
```

Der Inhalt des Kombinationsfeldes wird ausgelesen:

```
    stInhalt = oFeldList.getCurrentValue()
```

Der angezeigte Inhalt des Kombinationsfeldes wird ausgelesen. Leertasten und nicht druckbare Zeichen am Anfang und Ende der Eingabe werden gegebenenfalls entfernt.

```
    stInhalt = Trim(stInhalt)
    IF stInhalt <> "" THEN
        IF NameTabellenFeld2 <> "" THEN
```

Wenn ein zweites Tabellenfeld existiert, muss der Inhalt des Kombinationsfeldes gesplittet werden. Um zu wissen, an welcher Stelle die Aufteilung vorgenommen werden soll, ist der Feldtrenner von Bedeutung, der der Funktion als Variable mitgegeben wird. Ein Leerzeichen aus dem Feldtrenner- wird bei der Funktion «Split» nicht direkt erkannt. Deswegen wird das ASCII-Zeichen noch einmal in den entsprechenden Feldtrenner umgewandelt.

```
IF ASC(Feldtrenner) = 32 THEN
    Feldtrenner = " "
END IF
a_stTeile = Split(stInhalt, Feldtrenner, 2)
```

Der letzte Parameter weist darauf hin, dass maximal 2 Teile erzeugt werden.

Abhängig davon, welcher Eintrag mit dem Feld 1 und welcher mit dem Feld 2 zusammenhängt, wird jetzt der Inhalt des Kombinationsfeldes den einzelnen Variablen zugewiesen. «Position = 2» wird hier als Zeichen dafür genommen, dass an zweiter Position der Inhalt für das Feld 2 steht. Das ist auch dann der Fall, wenn beide Felder aus einer Tabelle stammen.

```
IF Position = 2 OR Position = 0 THEN
    stInhalt = Trim(a_stTeile(0))
    IF UBound(a_stTeile()) > 0 THEN
        stInhaltFeld2 = Trim(a_stTeile(1))
    ELSE
        stInhaltFeld2 = ""
    END IF
    stInhaltFeld2 = Trim(a_stTeile(1))
ELSE
    stInhaltFeld2 = Trim(a_stTeile(0))
    IF UBound(a_stTeile()) > 0 THEN
        stInhalt = Trim(a_stTeile(1))
    ELSE
        stInhalt = ""
    END IF
    stInhalt = Trim(a_stTeile(1))
END IF
END IF
```

Es kann passieren, dass bei zwei voneinander zu trennenden Inhalten die Größeneinstellung des Kombinationsfeldes (Textlänge) nicht zu einem der abzuspeichernden Tabellenfelder passt. Bei Kombinationsfeldern für nur ein Feld wird dies in der Regel durch Einstellungen im Formularkontrollfeld erledigt. Hier muss hingegen ein eventueller Fehler abgefangen werden. Es wird darauf hingewiesen, wie lang der Inhalt für das jeweilige Feld sein darf.

```
IF (LaengeFeld1 > 0 AND Len(stInhalt) > LaengeFeld1) OR
   (LaengeFeld2 > 0 AND Len(stInhaltFeld2) > LaengeFeld2) THEN
```

Wenn die Feldlänge des 1. oder 2. Teiles zu groß ist, wird erst einmal ein Standardtext in je einer Variablen abgespeichert. **`CHR(13)`** fügt hier einen Zeilenumbruch hinzu.

```
stmsgbox1 = "Das Feld " + NameTabellenFeld1 + " darf höchstens " +
   LaengeFeld1 + "Zeichen lang sein." + CHR(13)
stmsgbox2 = "Das Feld " + NameTabellenFeld2 + " darf höchstens " +
   LaengeFeld2 + "Zeichen lang sein." + CHR(13)
```

Sind beide Feldinhalte zu lang, so wird der Text mit beiden Feldinhalten ausgegeben.

```
IF (LaengeFeld1 > 0 AND Len(stInhalt) > LaengeFeld1) AND
   (LaengeFeld2 > 0 AND Len(stInhaltFeld2) > LaengeFeld2) THEN
   msgbox ("Der eingegebene Text ist zu lang." + CHR(13) +
      stmsgbox1 + stmsgbox2 + "Bitte den Text kürzen.",
      64,"Fehlerhafte Eingabe")
```

Die Anzeige erfolgt mit der Funktion 'msgbox()'. Sie erwartet zuerst einen Text, dann optional einen Zahlenwert (der zu einer entsprechenden Darstellungsform gehört) und schließlich optional einen Text als Überschrift über dem Fenster. Das Fenster hat hier also die Überschrift "Fehlerhafte Eingabe", die '64' fügt das Informationssymbol hinzu.

Im Folgenden werden alle auftretenden weiteren Fälle zu großer Textlänge abgearbeitet.

```
        ELSEIF (LaengeFeld1 > 0 AND Len(stInhalt) > LaengeFeld1) THEN
            msgbox ("Der eingegebene Text ist zu lang." + CHR(13) +
                stmsgbox1 + "Bitte den Text kürzen.",64,"Fehlerhafte Eingabe")
        ELSE
            msgbox ("Der eingegebene Text ist zu lang." + CHR(13) +
                stmsgbox2 + "Bitte den Text kürzen.",64,"Fehlerhafte Eingabe")
        END IF
    ELSE
```

Liegt kein zu langer Text vor, so kann die Funktion weiter durchlaufen. Ansonsten endet sie hier.

Jetzt werden die Inhaltseingaben so maskiert, dass eventuell vorhandene Hochkommata keine Fehlermeldung erzeugen.

```
        stInhalt = String_to_SQL(stInhalt)
        IF stInhaltFeld2 <> "" THEN
            stInhaltFeld2 = String_to_SQL(stInhaltFeld2)
        END IF
```

Zuerst werden Variablen vorbelegt, die anschließend per Abfrage geändert werden können. Die Variablen **`inID1`** und **`inID2`** sollen den Inhalt der Primärschlüsselfelder der beiden Tabellen speichern. Da bei einer Abfrage, die kein Ergebnis wiedergibt, durch Basic einer Integer-Variablen 0 zugewiesen wird, dies aber für das Abfrageergebnis auch bedeuten könnte, dass der ermittelte Primärschlüssel eben den Wert 0 hat, wird die Variable auf jeweils -1 voreingestellt. Diesen Wert nimmt ein Autowert-Feld bei der HSQLDB nicht automatisch an.

Anschließend wird die Datenbankverbindung erzeugt, soweit sie nicht schon besteht.

```
        inID1 = -1
        inID2 = -1
        oDatenquelle = ThisComponent.Parent.CurrentController
        If NOT (oDatenquelle.isConnected()) Then
            oDatenquelle.connect()
        End If
        oVerbindung = oDatenquelle.ActiveConnection()
        oSQL_Anweisung = oVerbindung.createStatement()
        IF NameTabellenFeld2 <> "" AND NOT IsEmpty(stInhaltFeld2) AND
            NameTabelle2 <> "" THEN
```

Wenn ein zweites Tabellenfeld existiert, muss zuerst die zweite Abhängigkeit geklärt werden. Zuerst wird überprüft, ob für den zweiten Wert in der Tabelle 2 bereits ein Eintrag existiert. Existiert dieser Eintrag nicht, so wird er eingefügt.

Beispiel: Die Tabelle 2 ist die Tabelle "Ort". In ihr werden also Orte abgespeichert. Ist z.B. ein Eintrag für den Ort 'Rheine' vorhanden, so wird der entsprechende Primärschlüsseleintrag ausgelesen. Ist der Eintrag 'Rheine' nicht vorhanden, wird er eingefügt und anschließend der beim Einfügen erzeugte Primärschlüsselwert festgestellt.

```
            stSql = "SELECT ""ID"" FROM """ + NameTabelle2 + """ WHERE """ +
                NameTabellenFeld2 + """='" + stInhaltFeld2 + "'"
            oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
            WHILE oAbfrageergebnis.next
                inID2 = oAbfrageergebnis.getInt(1)
            WEND
            IF inID2 = -1 THEN
                stSql = "INSERT INTO """ + NameTabelle2 + """ (""" +
                    NameTabellenFeld2 + """) VALUES ('" + stInhaltFeld2 + "') "
                oSQL_Anweisung.executeUpdate(stSql)
                stSql = "CALL IDENTITY()"
```

Ist der Inhalt in der entsprechenden Tabelle nicht vorhanden, so wird er eingefügt. Der dabei entstehende Primärschlüsselwert wird anschließend ausgelesen. Ist der Inhalt bereits vorhanden, so wird der Primärschlüsselwert durch die vorangehende Abfrage ermittelt. Die Funktion geht hier von automatisch erzeugten Primärschlüsselfeldern (**`IDENTITY`**) aus.

```
                oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
                WHILE oAbfrageergebnis.next
                    inID2 = oAbfrageergebnis.getInt(1)
                WEND
```

```
END IF
```

Der Primärschlüssel aus dem zweiten Wert wird in der Variablen **'inID2'** zwischengespeichert. Jetzt wird überprüft, ob eventuell dieser Schlüsselwert bereits in der Tabelle 1 zusammen mit dem Eintrag aus dem ersten Feld vorhanden ist. Ist diese Kombination nicht vorhanden, so wird sie neu eingefügt.

Beispiel: Für den Ort 'Rheine' aus der Tabelle 2 können in der Tabelle 1 mehrere Postleitzahlen verfügbar sein. Ist die Kombination '48431' und 'Rheine' vorhanden, so wird nur der Primärschlüssel aus der Tabelle 1 ausgelesen, in der die Postleitzahlen und der Fremdschlüssel aus der Tabelle 2 gespeichert wurden.

```
stSql = "SELECT ""ID"" FROM """ + NameTabelle1 + """ WHERE """ +
   NameTab12ID + """='" + inID2 + "' AND """ +
   NameTabellenFeld1 + """ = '" + stInhalt + "'"
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
WHILE oAbfrageergebnis.next
   inID1 = oAbfrageergebnis.getInt(1)
WEND
```

War der Inhalt der ersten Tabelle noch nicht vorhanden, so wird der Inhalt neu abgespeichert (**INSERT**).

Beispiel: Existiert bereits die Postleitzahl '48429' in Kombination mit dem Fremdschlüssel aus der Tabelle 2 "Ort", so wird auf jeden Fall ein neuer Datensatz erzeugt, wenn jetzt die Postleitzahl '48431' auftaucht. Der vorhergehenden Datensatz wird also nicht auf die neue Postleitzahl geändert. Schließlich sind durch die n:1-Verknüpfung der Tabellen mehrere Postleitzahlen für einen Ort ermöglicht worden.

```
IF inID1 = -1 THEN
   stSql = "INSERT INTO """ + NameTabelle1 + """ (""" +
      NameTabellenFeld1 + """,""" + NameTab12ID + """)
      VALUES ('" + stInhalt + "','" + inID2 + "') "
   oSQL_Anweisung.executeUpdate(stSql)
```

Der Primärschlüssel der ersten Tabelle muss schließlich wieder ausgelesen werden, damit er in die dem Formular zugrundeliegende Tabelle übertragen werden kann.

```
      stSql = "CALL IDENTITY()"
      oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
      WHILE oAbfrageergebnis.next
         inID1 = oAbfrageergebnis.getInt(1)
      WEND
   END IF
END IF
```

Für den Fall, dass beide in dem Kombinationsfeld zugrundeliegenden Felder in einer Tabelle gespeichert sind (z. B. Nachname, Vorname in der Tabelle Name) muss eine andere Abfrage erfolgen:

```
IF NameTabellenFeld2 <> "" AND NameTabelle2 = "" THEN
   stSql = "SELECT ""ID"" FROM """ + NameTabelle1 + """ WHERE """ +
      NameTabellenFeld1 + """='" + stInhalt + "' AND """ +
      NameTabellenFeld2 + """='" + stInhaltFeld2 + "'"
   oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
   WHILE oAbfrageergebnis.next
      inID1 = oAbfrageergebnis.getInt(1)
   WEND
   IF inID1 = -1 THEN
```

Wenn eine zweite Tabelle nicht existiert:

```
      stSql = "INSERT INTO """ + NameTabelle1 + """ (""" +
         NameTabellenFeld1 + """,""" + NameTabellenFeld2 + """)
         VALUES ('" + stInhalt + "','" + stInhaltFeld2 + "') "
      oSQL_Anweisung.executeUpdate(stSql)
```

Anschließend wird das Primärschlüsselfeld wieder ausgelesen.

```
      stSql = "CALL IDENTITY()"
```

```
                    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
                    WHILE oAbfrageergebnis.next
                       inID1 = oAbfrageergebnis.getInt(1)
                    WEND
                 END IF
              END IF
              IF NameTabellenFeld2 = "" THEN
```

Jetzt wird der Fall geklärt, der der einfachste ist: Das 2. Tabellenfeld existiert nicht und der Eintrag ist noch nicht in der Tabelle vorhanden. In das Kombinationsfeld ist also ein einzelner neuer Wert eingetragen worden.

```
                 stSql = "SELECT ""ID"" FROM """ + NameTabelle1 + """ WHERE """ +
                    NameTabellenFeld1 + """='" + stInhalt + "'"
                 oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
                 WHILE oAbfrageergebnis.next
                    inID1 = oAbfrageergebnis.getInt(1)
                 WEND
                 IF inID1 = -1 THEN
```

Wenn ein zweites Tabellenfeld nicht existiert, wird der Inhalt neu eingefügt ...

```
                    stSql = "INSERT INTO """ + NameTabelle1 + """ (""" +
                       NameTabellenFeld1 + """) VALUES ('" + stInhalt + "') "
                    oSQL_Anweisung.executeUpdate(stSql)
```

… und die entsprechende ID direkt wieder ausgelesen. (HSQLDB, ~~FIREBIRD~~)

```
                    stSql = "CALL IDENTITY()"
                    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
                    WHILE oAbfrageergebnis.next
                       inID1 = oAbfrageergebnis.getInt(1)
                    WEND
                 END IF
              END IF
```

Der Wert des Primärschlüsselfeldes muss ermittelt werden, damit er in die Haupttabelle des Formulars übertragen werden kann.

Anschließend wird der aus all diesen Schleifen ermittelte Primärschlüsselwert in das Feld der Haupttabelle und die darunterliegende Datenbank übertragen. Mit **`findColumn`** wird das mit dem Formularfeld verbundene Tabellenfeld erreicht. Mit **`updateLong`** wird eine Integer-Zahl (siehe «Datentypen ins StarBasic») diesem Feld zugewiesen.

```
                 oForm.updateLong(oForm.findColumn(oFeldList.Tag),inID1)
              END IF
           ELSE
```

Ist kein Primärschlüsselwert einzutragen, weil auch kein Eintrag in dem Kombinationsfeld erfolgte oder dieser Eintrag gelöscht wurde, so ist auch der Inhalt des Feldes zu löschen. Mit **`updateNULL()`** wird das Feld mit dem datenbankspezifischen Ausdruck für ein leeres Feld, **`NULL`**, versehen.

```
                 oForm.updateNULL(oForm.findColumn(oFeldList.Tag),NULL)
              END IF
           NEXT inCom
        END IF
     END SUB
```

Kontrollfunktion für die Zeichenlänge der Kombinationsfelder

Die folgende Funktion soll die Zeichenlänge der jeweiligen Tabellenspalten ermitteln, damit zu lange Eingaben nicht einfach gekürzt werden. Der Typ **`FUNCTION`** wurde hier wegen der Rückgabewerte gewählt.

```
FUNCTION Spaltengroesse(Tabellenname AS STRING, Feldname AS STRING) AS INTEGER
   oDatenquelle = ThisComponent.Parent.CurrentController
   If NOT (oDatenquelle.isConnected()) Then
      oDatenquelle.connect()
   End If
```

```
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "SELECT ""COLUMN_SIZE"" FROM ""INFORMATION_SCHEMA"".""SYSTEM_COLUMNS""
       WHERE ""TABLE_NAME"" = '" + Tabellenname + "' AND ""COLUMN_NAME"" = '"
       + Feldname + "'"
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
    WHILE oAbfrageergebnis.next
       i = oAbfrageergebnis.getInt(1)
    WEND
    Spaltengroesse = i
END FUNCTION
```

Hinweis	Für FIREBIRD muss der SQL-Code angepasst werden:

```
  stSql = "SELECT B.RDB$FIELD_LENGTH
     FROM RDB$RELATION_FIELDS AS A, RDB$FIELDS AS B
     WHERE A.RDB$FIELD_SOURCE = B.RDB$FIELD_NAME
        AND A.RDB$RELATION_NAME = '" + Tabellenname + "'
        AND A.RDB$FIELD_NAME = '" + Feldname + "'"
```

Datensatzaktion erzeugen

```
SUB Datensatzaktion_erzeugen(oEvent AS OBJECT)
```

Dieses Makro sollte an das folgende Ereignis des Listenfeldes gebunden werden: 'Bei Fokuserhalt'. Es ist notwendig, damit auf jeden Fall bei einer Änderung des Listenfeldinhaltes die Speicherung abläuft. Ohne dieses Makro wird keine Änderung in der Tabelle erzeugt, die für Base wahrnehmbar ist, da die Combobox mit dem Formular nicht verbunden ist.

Dieses Makro stellt direkt die Eigenschaft des Formulars um.

```
    DIM oForm AS OBJECT
    oForm = oEvent.Source.Model.Parent
    oForm.IsModified = TRUE
END SUB
```

Bei Formularen, die bereits ihren Inhalt auch für die Kombinationsfelder aus Abfragen erhalten, ist dieses Makro nicht notwendig. Änderungen in den Kombinationsfeldern werden direkt registriert.

Navigation von einem Formular zum anderen

Ein Formular soll über ein entsprechendes Ereignis geöffnet werden.

Im Formularkontrollfeld wird unter den Eigenschaften in der Zeile "Zusatzinformationen" (Tag) hier der Name des Formulars eintragen. Hier können auch weitere Informationen eingetragen werden, die über den Befehl **`Split()`** anschließend voneinander getrennt werden.

```
SUB Zu_Formular_von_Formular(oEvent AS OBJECT)
    DIM stTag AS String
    stTag = oEvent.Source.Model.Tag
    aForm() = Split(stTag, ",")
```

Das Array wird gegründet und mit den Formularnamen gefüllt, in diesem Fall zuerst in dem zu öffnenden Formular und als zweites dem aktuellen Formular, dass nach dem Öffnen des anderen geschlossen werden soll. Existiert ein zweiter Eintrag nicht, so wird durch das Makro nur ein neues Formular geöffnet.

```
    ThisDatabaseDocument.FormDocuments.getByName( Trim(aForm(0)) ).open
    IF Ubound(aForm()) > 0 THEN
       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForm(1)) ).close
    END IF
END SUB
```

Soll stattdessen nur beim Schließen ein anderes Formular geöffnet werden, weil z.B. ein Hauptformular existiert und alle anderen Formulare von diesem aus über entsprechende Buttons angesteu-

ert werden, so ist das folgende Makro einfach an das Formular unter **Extras → Anpassen → Ereignisse → Dokument wird geschlossen** anzubinden:

```
SUB Hauptformular_oeffnen
    ThisDatabaseDocument.FormDocuments.getByName( "Hauptformular" ).open
END SUB
```

Wenn die Formulardokumente innerhalb der *.odb-Datei in Verzeichnissen sortiert sind, so muss das Makro für den Formularwechsel etwas umfangreicher sein:

```
SUB Zu_Formular_von_Formular_mit_Ordner(oEvent AS OBJECT)
    REM Das zu öffenende Formular wird als erstes angegeben.
    REM Liegt ein Formular in einem Ordner, so ist die Beziehung über "/" zu
       definieren,
    REM so dass der Unterordner zu finden ist.
    DIM stTag AS STRING
    stTag = oEvent.Source.Model.Tag 'Tag wird unter den Zusatzinformationen eingegeben
    aForms() = Split(stTag, ",")  'Hier steht zuerst der Formularname für das neue
       Formular, dann der für das alte Formular
    aForms1() = Split(aForms(0),"/")
    aForms2() = Split(aForms(1),"/")
    IF UBound(aForms1()) = 0 THEN
       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms1(0)) ).open
    ELSE
       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms1(0)) ).getByName(
          Trim(aForms1(1)) ).open
    END IF
    IF UBound(aForms2()) = 0 THEN
       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms2(0)) ).close
    ELSE
       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms2(0)) ).getByName(
          Trim(aForms2(1)) ).close
    END IF
END SUB
```

Formulardokumente, die in einem Verzeichnis liegen, werden in den Zusatzinformationen als Verzeichnis/Formular angegeben. Dies muss umgewandelt werden zu **`...getByName("Verzeichnis").getByName("Formular")`**.

Tabellen, Abfragen, Formulare und Berichte öffnen

Ähnlich wie im vorhergehenden Kapitel lassen sich von einem Formular aus auch Berichte öffnen. Berichte sind wie Formulare in die Base-Datei eingebundene separate Dokumente. Statt **`FormDocuments`** ist hier lediglich **`ReportDocuments`** einzutragen. Außerdem ist noch darauf zu achten, dass sowohl Formulare als auch Berichte in Unterverzeichnissen liegen können. Schwieriger ist es hingegen, auch auf Tabellen, Abfragen und Ansichten zuzugreifen, da diese nicht als separate Dokumente vorliegen.

```
SUB Navigation(oEvent AS OBJECT)
    DIM stTag AS STRING
    DIM inType AS INTEGER
    stTag = oEvent.Source.Model.Tag
    aOpen() = Split(stTag, ",")
    SELECT CASE Trim(aOpen(0))
       CASE "form", "report"
          REM Forms and Reports could be saved also in subfolders.
          aForms1() = Split(Trim(aOpen(1)),"/")
          IF Trim(aOpen(0)) = "form" THEN
             oDoc = ThisDatabaseDocument.FormDocuments
          ELSE
             oDoc = ThisDatabaseDocument.ReportDocuments
          END IF
          IF Ubound(aForms1()) > 0 THEN
             oDoc.getByName( Trim(aForms1(0)) ).getByName( Trim(aForms1(1)) ).open
          ELSE
```

```
                oDoc.getByName( Trim(aForms1(0)) ).open
            END IF
            IF Trim(aOpen(0)) = "form" AND Ubound(aOpen()) > 1 THEN
                REM The Form, which starts the Macro, could also be closed ...
                aForms2() = Split(Trim(aOpen(2)),"/")
                IF Ubound(aForms2()) > 0 THEN
                ThisDatabaseDocument.FormDocuments.
                        getByName( Trim(aForms2(0)) ).getByName( Trim(aForms2(1)) ).close
                ELSE
                ThisDatabaseDocument.FormDocuments.
                        getByName( Trim(aForms2(0)) ).close
                END IF
            END IF
            EXIT SUB
        CASE "query"
            inType = 1
            Open_Table_Query_View(Trim(aOpen(1)),inType)
        CASE "table"
            inType = 0
            Open_Table_Query_View(Trim(aOpen(1)),inType)
    END SELECT
END SUB
```

Über die Prozedur Navigation wird das Makro gestartet. Von den Buttons wird aus den Zusatzinformationen (**`Tag`**) die Information ausgelesen, ob ein Formular (**`form`**), ein Bericht (**`report`**) usw. aufgerufen werden soll. Der Name des Formulars, Berichtes usw. wird in den Zusatzinformationen durch ein Komma von dieser Information getrennt.

Enthält der erste Teil des daraus ermittelten Arrays die Bezeichnung **`form`**, so wird anschließend das Formular geöffnet. Entsprechendes gilt für die Bezeichnung **`report`**, die den **`SELECT CASE`** für den Bericht ergibt.

Für Abfragen und Tabellen muss ein anderer Weg beschritten werden. Hier wird sowohl der Name der Abfrage bzw. Tabelle als auch eine Integer-Zahl an die folgende Prozedur **`Open_Table_Query_View`** weitergegeben.

```
SUB Open_Table_Query_View(stName AS STRING, inType AS INTEGER)
    DIM oController AS OBJECT
    DIM oConnection AS OBJECT
    oController = ThisDatabasedocument.CurrentController
    IF NOT oController.isconnected THEN oController.connect
    oConnection = oController.ActiveConnection
    DIM URL AS NEW com.sun.star.util.URL
    DIM Args(5) AS NEW com.sun.star.beans.PropertyValue
    URL.Complete = ".component:DB/DataSourceBrowser"
    Dispatch = StarDesktop.queryDispatch(URL,"_Blank",8)
    Args(0).Name = "ActiveConnection"
    Args(0).Value = oConnection
    Args(1).Name = "CommandType"
    Args(1).Value = inType   '0=Table 1=SQL_Query 2=Command
    Args(2).Name = "Command"
    Args(2).Value = stName
    Args(3).Name = "ShowMenu"
    Args(3).Value = True
    Args(4).Name = "ShowTreeView"
    Args(4).Value = False
    Args(5).Name = "ShowTreeViewButton"
    Args(5).Value = False
    Dispatch.dispatch(URL, Args)
END SUB
```

Zuerst wird die Verbindung zur Datenbank hergestellt, sofern sie noch nicht existiert. Diese Verbindung muss mit einigen zusätzlichen Informationen, unter anderem der Art des zu öffnenden Elementes (Tabelle oder Abfrage) sowie dem Namen des Elementes, in einem Array weiter gegeben werden.

Die Tabelle bzw. Abfrage wird schließlich über den **`queryDispatch`** mit dem Kommando **`dispatch`** geöffnet.

Hierarchische Listenfelder

Einstellungen in einem Listenfeld sollen die Einstellungen in einem zweiten Listenfeld direkt beeinflussen. Auf einfachere Art und Weise wurde dies schon bei der Filterung von Datensätzen weiter oben beschrieben. Jetzt soll aber hinzu kommen, dass das erste Listenfeld den Inhalt des zweiten Listenfeldes beeinflusst, der wiederum den Inhalt des dritten Listenfeldes beeinflusst usw.

Jahrgang
1
2
3
4
5
6
7
8
9
10
11
12
13

Klasse
a
b
c
d
e
f
g

Name
Karl Müller
Evelyn Maier
Maria Gott
Eduard Abgefahren
Kurt Drechsler
Kunigunde Schimmel

Abbildung 1: Beispielhafte Listenfelder für eine hierarchische Anordnung von Listenfeldern.

In diesem Beispiel enthält Listenfeld 1 alle Jahrgänge der Schule. Die Klassen der jeweiligen Jahrgänge sind durch Buchstaben kenntlich gemacht. Die Namen enthalten die Schülerinnen und Schüler der Klasse.

Unter normalen Umständen zeigt das Listenfeld für den Jahrgang alle 13 Jahrgänge an. Das Listenfeld für die Klasse alle Buchstaben und das Listenfeld für die Schüler und Schülerinnen alle Schüler und Schülerinnen der Schule.

Wird mit hierarchischen Listenfeldern gearbeitet, so wird nach Auswahl des Jahrganges das Listenfeld für die Klasse eingegrenzt. Es werden nur noch die Klassenbezeichnungen angezeigt, die es in dem Jahrgang tatsächlich gibt. So könnte eben bei steigender Schüler- und Schülerinnenzahl die Anzahl der Klassen im Jahrgang ebenfalls steigen. Das letzte Listenfeld, die Namen, ist jetzt bereits stark eingegrenzt. Statt alle vermutlich deutlich über 1000 Schüler und Schülerinnen anzuzeigen, listet das letzte Feld nur noch die ca. 30 Schüler und Schülerinnen der einen letztlich ausgewählten Klasse auf.

Zum Beginn steht nur die Auswahl des Jahrganges zur Verfügung. Ist ein Jahrgang ausgewählt, so steht die (bereits eingeschränkte) Auswahl der Klasse zur Verfügung. Erst zum Schluss wird schließlich das Listenfeld für die Namen freigegeben.

Wird das Listenfeld des Jahrganges geändert, so muss der Durchlauf wieder wie vorher starten. Wird nur das Listenfeld der Klasse geändert, so muss der Wert des Jahrganges für das letzte Listenfeld der Namen weiter gelten.

Um solch eine Funktion bereitzustellen, muss innerhalb eines Formulars eine Variable zwischengespeichert werden. Dies erfolgt in einem versteckten Kontrollfeld.

Der Makrostart wird an die Veränderung des Inhaltes eines Listenfeldes gekoppelt: **Eigenschaft Listenfeld → Ereignisse → Modifiziert.** In den Zusatzinformationen des Listenfeldes werden die notwendige Variablen gespeichert.

Hier der beispielhafte Inhalt der Zusatzinformationen:
`Jahrgang,verstecktes Kontrollfeld,Listenfeld 2`

Das aktuelle Listenfeld ist als «Listenfeld 1» bezeichnet. Dieses Listenfeld stellt den Inhalt des Tabellenfeldes «Jahrgang» dar. Nach diesem Eintrag muss also das darauffolgende Listenfeld gefiltert werden. Das versteckte Kontrollfeld ist in diesem Fall auch gleich mit dem entsprechenden Namen gekennzeichnet. Und schließlich wird noch darauf hingewiesen, dass ein 2. Listenfeld, «Listenfeld 2», existiert, an das die Filterung weiter gegeben wird.

```
SUB Hierarchisches_Kontrollfeld(oEvent AS OBJECT)
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeldHidden AS OBJECT
    DIM oFeld AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM stSql AS STRING
    DIM aInhalt()
    DIM stTag AS STRING
    oFeld = oEvent.Source.Model
    stTag = oFeld.Tag
    oForm = oFeld.Parent
    REM Tag wird unter den Zusatzinformationen eingegeben
    REM Hier steht:
    REM 0. Feldname des zu filterndes Feldes in der Tabelle,
    REM 1. Feldname des versteckten Konrollfeldes, das den Filterwert speichern soll,
    REM 2. eventuell weiteres Listfeld
    REM Der Tag wird von dem auslösenden Element ausgelesen. Die Variable wird an die
       Prozedur weitergegeben, die gegebenenfalls alle weiteren Listenfelder
       einstellt.
    aFilter() = Split(stTag, ",")
    stFilter = ""
```

Nachdem die Variablen deklariert wurden, wird der Inhalt des Tags in ein Array übertragen. So kann auf die einzelnen Elemente zugegriffen werden. Anschließend wird der Zugang zu den verschiedenen Feldern im Formular deklariert.

Das Listenfeld wird aus dem Aufruf heraus ermittelt. Aus dem Listenfeld wird der Wert ausgelesen. Nur wenn dieser Wert einen Inhalt hat, wird er mit dem Feldnamen des zu filternden Feldes, in unserem Beispiel «Jahrgang», zu einer SQL-Bedingung kombiniert. Ansonsten bleibt der Filter leer. Sind die Listenfelder zur Filterung eines Formulars gedacht, dann ist kein verstecktes Kontrollfeld vorhanden. Unter dieser Bedingung wird der Filterwert direkt im Formular gespeichert.

```
    IF Trim(aFilter(1)) = "" THEN
        IF oFeld.getCurrentValue <> "" THEN
            stFilter = """"+Trim(aFilter(0))+"""='"+oFeld.getCurrentValue()+"'"
```

Existiert bereits vorher ein Filter (weil es sich z.B. um das Listenfeld 2 handelt, das jetzt betätigt wurde), so wird der neue Inhalt an den vorherigen angehängt, der in dem versteckten Feld zwischengespeichert wurde.

```
            IF oForm.Filter <> ""
```

Dies darf allerdings nur dann geschehen, wenn das gleiche Feld noch nicht gefiltert wurde. Schließlich ist z.B. bei einer Filterung nach dem «Jahrgang» kein Datensatz unter «Name» mehr

zu erwarten, wenn zusätzlich eine weitere Filterung nach «Jahrgang» erfolgt. Eine Person kann immer nur in einem «Jahrgang» existieren. Es muss also ausgeschlossen werden, dass in der Filterung der Filtername bereits vorkommt.

```
            AND InStr(oForm.Filter, """"+Trim(aFilter(0))+"""='") = 0 THEN
            stFilter = oForm.Filter + " AND " + stFilter
```

Existiert bereits ein Filter und kommt das Feld, nach dem gefiltert werden soll, bereits im Filter vor, so muss die vorherige Filterung ab diesem Feldnamen gelöscht und die neue Filterung eingefügt werden.

```
        ELSEIF oForm.Filter <> "" THEN
            stFilter = Left(oForm.Filter,
            InStr(oForm.Filter, """"+Trim(aFilter(0))+"""='")-1) + stFilter
        END IF
    END IF
```

Anschließend wird der Filter in das Formular eingetragen. Dieser Filter kann auch leer sein, wenn direkt das erste Listenfeld ohne Inhalt gewählt wurde.

```
    oForm.Filter = stFilter
    oForm.reload()
```

Die gleiche Prozedur wird durchlaufen, wenn nicht ein Formular direkt gefiltert werden soll. In dem Fall wird der Filterwert in einem versteckten Kontrollfeld zwischengespeichert.

```
ELSE
    oFeldHidden = oForm.getByName(Trim(aFilter(1)))
    IF oFeld.getCurrentValue <>"" THEN
        stFilter = """"+Trim(aFilter(0))+"""='"+oFeld.getCurrentValue()+"'"
        IF oFeldHidden.HiddenValue <> ""
        AND InStr(oFeldHidden.HiddenValue, """"+Trim(aFilter(0))+"""='") = 0 THEN
            stFilter = oFeldHidden.HiddenValue + " AND " + stFilter
        ELSEIF oFeldHidden.HiddenValue <> "" THEN
            stFilter = Left(oFeldHidden.HiddenValue,
                InStr(oFeldHidden.HiddenValue, """"+Trim(aFilter(0))+"""='")-1) +
                stFilter
        END IF
    END IF
    oFeldHidden.HiddenValue = stFilter
END IF
```

Ist in den Zusatzinformationen ein 4. Eintrag (Arraynummerierung beginnt bei 0!) vorhanden, so muss das folgende Listenfeld jetzt auf den entsprechenden Eintrag des aufrufenden Listenfeldes eingestellt werden.

```
IF UBound(aFilter()) > 1 THEN
    oFeld1 = oForm.getByName(Trim(aFilter(2)))
    aFilter1() = Split(oFeld1.Tag,",")
```

Die notwendigen Daten für die Filterung werden aus den Zusatzinformationen («Tag») des entsprechenden Listenfeldes ausgelesen. Leider ist es nicht möglich, lediglich den SQL-Code in dem Listenfeld neu zu schreiben und anschließend das Listenfeld einzulesen. Vielmehr müssen die entsprechenden Werte direkt in das Listenfeld geschrieben werden.

Bei der Erstellung des Codes wird davon ausgegangen, dass die Tabelle, auf der das Formular beruht, die gleiche ist, auf der auch die Listenfelder beruhen. Für eine Weitergabe von Fremdschlüsseln an die Tabelle ist so ein Listenfeld also erst einmal nicht gedacht.

```
    IF oFeld.getCurrentValue <> "" THEN
        stSql = "SELECT DISTINCT """+Trim(aFilter1(0))+""" FROM """+oForm.Command+
            """ WHERE "+stFilter+" ORDER BY """+Trim(aFilter1(0))+""""
        oDatenquelle = ThisComponent.Parent.CurrentController
        If NOT (oDatenquelle.isConnected()) THEN
            oDatenquelle.connect()
        END IF
        oVerbindung = oDatenquelle.ActiveConnection()
        oSQL_Anweisung = oVerbindung.createStatement()
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

Die Werte werden in ein Array eingelesen. Das Array wird anschließend direkt in das Listenfeld übertragen. Die entsprechenden Zähler für das Array werden durch die Schleife kontinuierlich erhöht.

```
            inZaehler = 0
            WHILE oAbfrageergebnis.next
                ReDim Preserve aInhalt(inZaehler)
                aInhalt(inZaehler) = oAbfrageergebnis.getString(1)
                inZaehler = inZaehler+1
            WEND
        ELSE
            aInhalt(0) = ""
        END IF
        oFeld1.StringItemList = aInhalt()
```

Der Inhalte des Listenfeldes wurde neu erstellt. Das Listenfeld muss neu eingelesen werden. Anschließend wird anhand der Zusatzinformationen des neu eingestellten Listenfeldes jedes eventuell weiter folgende Listenfeld entsprechend geleert, indem eine Schleife für alle folgenden Listenfelder gestartet wird, bis eben ein letztes Listenfeld keinen 4. Eintrag in den Zusatzinformationen enthält.

```
        oFeld1.refresh()
        WHILE UBound(aFilter1()) > 1
            DIM aLeer()
            oFeld2 = oForm.getByName(Trim(aFilter1(2)))
            DIM aFilter1()
            aFilter1() = Split(oFeld2.Tag,",")
            oFeld2.StringItemList = aLeer()
            oFeld2.refresh()
        WEND
    END IF
END SUB
```

Die sichtbaren Inhalte des Listenfeldes werden in `oFeld1.StringItemList` gespeichert. Soll zusätzlich auch ein Wert gespeichert werden, der als Fremdschlüssel an die darunterliegende Tabelle weitergegeben wird, wie bei Listenfeldern in Formularen üblich, so ist dieser Wert in der Abfrage zusätzlich zu ermitteln und anschließend mit `oFeld1.ValueItemList` abzuspeichern.

Für so eine Erweiterung sind allerdings zusätzliche Variablen notwendig wie z.B. neben der Tabelle, in der die Werte des Formulars gespeichert werden, noch die Tabelle, aus der die Listenfeldinhalte gelesen werden.

Besondere Aufmerksamkeit ist dabei der Formulierung des Filters zu widmen.

```
stFilter = """"+Trim(aFilter(1))+"""='"+oFeld.getCurrentValue()+"'"
```

funktioniert dann nur noch, wenn es sich bei der zugrundeliegenden LO-Version um eine Version ab LO 4.1 handelt, da hier als `CurrentValue()` der Wert wiedergegeben wird, der auch abgespeichert wird – nicht der Wert, der lediglich angezeigt wird. Damit das einwandfrei über verschiedene Versionen hinweg funktioniert, sollte unter **Eigenschaften: Listenfeld → Daten → Gebundenes Feld → '0'** angegeben sein.

Zeiteingaben mit Millisekunden

Um Zeiten im Millisekunden-Bereich zu speichern, ist in der Tabelle ein Timestamp-Feld erforderlich, das zudem per SQL separat darauf eingestellt wird (siehe «Zeitfelder in Tabellen») (HSQLDB, FIREBIRD erlaubt auch Millisekunden für normale Zeiten). Ein solches Feld kann vom Formular aus mit einem formatierten Feld beschrieben werden, das auch das Format **`MM:SS,00`** anbietet. Allerdings scheitert der erste Schreibversuch daran, dass der Eingabe der entsprechende Datumszusatz fehlt. Dies kann mit dem folgenden Makro erreicht werden, das an die Eigenschaft «Vor der Datensatzaktion» des Formulars gebunden wird:

```
SUB Timestamp
    DIM unoStmp AS NEW com.sun.star.util.DateTime
    DIM oDoc AS OBJECT
```

```
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld AS OBJECT
    DIM stZeit AS STRING
    DIM ar()
    DIM arMandS()
    DIM loNano AS LONG
    DIM inSecond AS INTEGER
    DIM inMinute AS INTEGER
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("MainForm")
    oFeld = oForm.getByName("Zeit")
    stZeit = oFeld.Text
```

Die Variablen werden vorher deklariert. Nur wenn das Feld «Zeit» einen Inhalt hat, wird der weitere Code ausgeführt. Sonst tritt der Mechanismus des Formulars in Kraft, der das Feld auf **`NULL`** setzt.

```
    IF stZeit <> "" THEN
        ar() = Split(stZeit,",")
        loNano = CLng(ar(1)&"0000000")
        arMandS() = Split(ar(0),":")
        inSecond = CInt(arMandS(1))
        inMinute = Cint(arMandS(0))
```

Die Einträge aus dem Feld «Zeit» werden in ihre Bestandteile zerlegt.

Zuerst werden die Hundertstelsekunden abgetrennt und mit so vielen Nullen rechts aufgefüllt, dass sich insgesamt eine neunstellige Zahl ergibt. Eine so hohe Zahl kann nur in einer Long-Variablen gespeichert werden.

Anschließend werden aus dem verbleibenden Rest durch eine Trennung am Trennzeichen «:» die Minuten von den Sekunden getrennt und in Integer-Zahlen umgewandelt.

```
        WITH unoStmp
            .NanoSeconds = loNano
            .Seconds = inSecond
            .Minutes = inMinute
            .Hours = 0
            .Day = 30
            .Month = 12
            .Year = 1899
        END WITH
```

Dem Zeitstempel wird nun das Standarddatum 30.12.1899 zugewiesen, das dem Standard-Startdatum von LibreOffice entspricht. Hier kann natürlich auch das aktuelle Datum mitgespeichert werden.

Hinweis

Aktuelles Datum ermitteln und speichern:

```
  DIM Jetzt AS DATE
      Jetzt = Now()
      WITH unoStmp
          .NanoSeconds = loNano
          .Seconds = inSecond
          .Minutes = inMinute
          .Hours = Hour(Jetzt)
          .Day = Day(Jetzt)
          .Month = Month(Jetzt)
          .Year = Year(Jetzt)
      END WITH
```

```
        oFeld.BoundField.updateTimestamp(unoStmp)
    END IF
END SUB
```

Anschließend wird der erzeugte Zeitstempel über **`updateTimestamp`** in das Feld übertragen und mit dem Formular abgespeichert.

In älteren Anleitungen wird hier statt **NanoSeconds** der Begriff **HundrethSeconds** verwendet. Dieser entspricht aber nicht der API von LibreOffice und erzeugt deshalb nur Fehlermeldungen.

Ein Ereignis – mehrere Implementationen

Bei Formularen kommt es vor, dass ein Makro, mit einem Ereignis verknüpft, gleich zweimal ausgeführt wird. Dies liegt daran, dass mehrere Prozesse gleichzeitig z.B. mit dem Abspeichern eines geänderten Datensatzes verbunden sind. Die unterschiedlichen Ursachen für so ein Ereignis lassen sich folgendermaßen ermitteln:

```
SUB Ereignisursache_ermitteln(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    oForm = oEvent.Source
    MsgBox oForm.ImplementationName
END SUB
```

Beim Abspeichern eines geänderten Datensatzes ergeben sich so zwei Implementationsnamen: **org.openoffice.comp.svx.FormController** und **com.sun.star.comp.forms.ODatabaseForm**. Über diese Namen kann jetzt gesteuert werden, dass ein Makro letztlich nur einmal den ganzen Code durchläuft. Die doppelte Durchführung ist oft nur eine (kleine) Bremse im Programmablauf. Sie kann aber auch dazu führen, dass sich z.B. ein Cursor nicht nur einen, sondern gleich zwei Datensätze zurück bewegt. Die Implementationen lassen auch nur bestimmte Befehle zu, so dass eine Kenntnis des Namens der Implementation von Bedeutung sein kann.

Eingabekontrolle bei Formularen

Ein Formular sollte für die Eingabe so weit wie möglich abgesichert sein, bevor die Daten in die Datenbank geschrieben werden. Dies erfolgt natürlich schon allein dadurch, dass Felder des Formulars passend zu den Inhalten aus der Datenbank gewählt werden. Auch lassen sich Felder so einstellen, dass sie eine zwingende Eingabe benötigen. Diese zwingende Eingabe muss zur Zeit (LO 6.1) allerdings auch in der Tabelle der Datenbank definiert sein. Die diesem Abschnitt zugrundeliegende Datenbank[10] zeigt fehlende Eingabe direkt an und vermeidet in einigen Feldern auch eventuell fehlerhafte Eingaben.

10 Siehe hierzu die Beispieldatenbank «Beispiel_Formular_Eingabekontrolle.odb»

Mehrere Elemente des Formulars fallen sofort auf:

- Das Feld «ID» ist mit einem separaten Hintergrund und einer separaten Umrandung versehen. Es ist über **Eigenschaften → Nur lesen** von der Eingabe ausgeschlossen. Außerdem ist **Tabstop → Nein** gesetzt. Hierzu ist kein Makroeinsatz erforderlich.
- Die Felder «Nachname», «Firma», «Straße», «PLZ – Ort» und «Foto» sind rot umrandet. Hier ist eine Eingabe erforderlich. Die Umrandung verschwindet sobald dort ein Text eingetragen wird.
- Das Feld «Keine Adresse» kann hier genutzt werden um die Felder «Straße» und «PLZ – Ort» zu deaktivieren. Die rote Umrandung verschwindet dann, die Hintergrundfarbe wird geändert und eine Eingabe ist nicht mehr möglich.
- Das Feld «Website» ist mit einem zusätzlichen Listenfeld versehen. Dies soll sicherstellen, dass die Eingabe mit 'http://' oder 'https://' beginnt. Die entsprechende Vorwahl steht bereits in dem Eingabefeld und wird durch die weitere Eingabe ergänzt.
- Bei dem Feld «Website» fällt bereits auf, dass in dem Feld der Text in blauer Farbe mit einfacher Unterstreichung abgebildet wird. Das Feld enthält einen Link, der bei gedrückter Strg-Taste mit der Maustaste angeklickt und geöffnet werden kann. Die entsprechende Funktion ist auch bei den Feldern «Email» und «Foto» (zur Großdarstellung des Fotos) hinterlegt.
- Das Feld «IBAN» ist ein Maskiertes Feld, das nur die Eingabe von Zahlen erlaubt. Hier erfolgt nach dem Verlassen des Feldes eine entsprechende Überprüfung auf korrekte Eingabe. Ähnliche Funktionen sind bei den Feldern «EMail» und «Website» hinterlegt.
- Der Button «Adresse auf Karte» schließlich öffnet eine Karte, auf der die eingegebene Adresse im Webbrowser angezeigt wird, sofern sie wirklich existiert und bei OpenStreetmap verzeichnet ist.

Erforderliche Eingaben absichern

Zu beginn werden einige globale Variablen festgelegt. Die Standardfarbe für den Rahmen und den Hintergrund eines Feldes muss verfügbar sein, ebenso die Farbe, in der der Rahmen erscheinen soll, wenn eine Eingabe notwendig ist. Alle Formularfelder, bei denen zum Start des Formulars **Daten → Eingabe erforderlich → Ja** eingestellt ist, werden in einem zentralen Array gespeichert. Ohne diese Speicherung wäre es nicht möglich, die erforderliche Eingabe z.B. für die Adresse ein- und wieder auszuschalten.

```
GLOBAL loBorderDefault AS LONG
GLOBAL loBorderInputRequired AS LONG
GLOBAL loColorStandard AS LONG
GLOBAL arFormInputRequired()
```

Die globalen Variablen werden beim Öffnen des Formulardokumentes mit Inhalt versehen. Dies regelt die Prozedur «FormVars».

Die Farbvariablen werden direkt festgelegt. Anschließend wird das gesamte Formular durchgegangen und alle Felder einzeln untersucht. Nur die Felder, die zu dem **`DataAwareControlModel`** gehören können auch Daten aufnehmen. Andere Felder wie Beschriftungsfelder, Buttons oder versteckte Felder können nicht für eine Eingabe genutzt werden.

Jetzt kann noch vorkommen, dass bei einem Feld zwar die Eingabe notwendig ist, leider aber keine Umrandungsfarbe einstellbar ist. Deswegen werden schließlich in das Array für die als notwendig zu versehenden Eingaben nur die übernommen, die auch die Eigenschaft **`BorderColor`** unterstützen.

```
SUB FormVars(oEvent AS OBJECT)
    DIM oForm AS OBJECT, oField AS OBJECT
    DIM k AS INTEGER, i AS INTEGER
    loBorderDefault = RGB(192,192,192) 'Grau
    loBorderInputRequired = RGB(255,0,0) 'Rot
    loColorStandard = RGB(250,250,250) 'sehr helles Grau
    oForm = oEvent.Source
    FOR i = 0 TO oForm.Count - 1
        oField = oForm.getByIndex(i)
        IF oField.supportsService("com.sun.star.form.DataAwareControlModel") THEN
            IF oField.InputRequired THEN
                IF oField.getPropertySetInfo.hasPropertyByName("BorderColor") THEN
                    REDIM PRESERVE arFormInputRequired(k)
                    arFormInputRequired(k) = oField.Name
                    k = k + 1
                END IF
            END IF
        END IF
    NEXT
    FormChange(oEvent)
END SUB
```

In der vorhergehenden Prozedur wird bereits die Prozedur «FormChange» aufgerufen. Mit dieser Prozedur wird die Kennzeichnung der notwendigen Eingaben vorgenommen.

```
SUB FormChange(oEvent AS OBJECT)
    DIM oForm AS OBJECT, oField AS OBJECT
    DIM i AS INTEGER, n AS INTEGER, k AS INTEGER
    DIM stTest AS STRING
    DIM a(), aa(), ab()
    oForm = oEvent.Source
```

In der ersten Schleife durch das Array der Formularfelder, bei denen eine Eingabe nötig ist, wird überprüft, ob das Feld einen Wert enthält. Hier muss zwischen Feldern unterschieden werden, die eine Verbindung zur Datenbank haben und solchen, die ohne Verbindung zur Datenbank existieren (Kombinationsfeld, für das der Fremdschlüssel über Makro ermittelt wird). Die einfache Abfrage nach **`CurrentValue`** führt bei Bildfeldern zu einem Fehler, weil dort diese Eigenschaft nicht existiert. Ist dies nicht der Fall, dann wird rot umrandet. Ist dies der Fall, dann wird die Standardumrandung gewählt.

```
    FOR i = LBound(arFormInputRequired()) TO UBound(arFormInputRequired())
        oField = oForm.getByName(arFormInputRequired(i))
        oField.InputRequired = True
        IF NOT IsNULL(oField.BoundField) THEN
            IF oField.BoundField.String = "" THEN
                oField.BorderColor = loBorderInputRequired
            ELSE
                oField.BorderColor = loBorderDefault
            END IF
        ELSEIF oField.CurrentValue = "" THEN
            oField.BorderColor = loBorderInputRequired
        ELSE
            oField.BorderColor = loBorderDefault
        END IF
    NEXT
```

Die darauffolgende zweite Schleife ist nur deswegen notwendig, weil das Formular ein Feld enthält, das die Eingabe für die Adresse ausschließt. In Abhängigkeit von diesem Feld muss also noch einmal überprüft werden, welche Felder denn jetzt eine Eingabe erfordern und mit einer roten Umrandung gezeigt werden müssen.

```
    FOR i = LBound(arFormInputRequired()) TO UBound(arFormInputRequired())
        oField = oForm.getByName(arFormInputRequired(i))
        IF NOT IsNULL(oField.BoundField) THEN
            stTest = oField.BoundField.String
        ELSE
            stTest = oField.CurrentValue
        END IF
        IF stTest <> "" AND oField.Tag <> "" THEN
```

In den Feldern, für die eine Eingabe notwendig ist, wird vermerkt von welchem Feld diese Eingabe abhängt. In der Beispieldatenbank steht in den Zusatzinformationen von «Nachname» **`notrequired[txtFirma]`**. Das soll bedeuten: Ist ein Nachname eingetragen, so ist bei der Firma kein Eintrag mehr notwendig. Entsprechend steht in den Zusatzinformationen von «Firma» **`notrequired[txtNachname]`**. Auch für andere Bereiche wurde in der Beispieldatenbank nach einem Kennwort eine Liste der entsprechenden Felder in eckigen Klammern gewählt. In den Zusatzinformationen können so mehrere Kennworte mit entsprechenden Listen untergebracht werden. Die abschließende eckige Klammer ist der Trenner, nach dem jetzt zuerst einmal gesucht wird:

```
            a = split(oField.Tag,"]")
            FOR n = LBound(a()) TO Ubound(a())-1
```

Da die Abschließende Klammer auch am Ende aller Eintragungen steht ist das letzte Arrayelement auf jeden Fall leer. Die Schleife muss also nur bis zum vorletzten Arrayelement laufen.

Enthält das Arrayelement den Begriff 'notrequired', so wird hier jetzt weiter nach den enthaltenen Felder gesucht. Zuerst wird mit Hilfe der geöffneten eckigen Klammer das Kennwort von den Feldbezeichnungen getrennt, dann werden die Feldbezeichnungen getrennt, sofern überhaupt innerhalb der eckigen Klammern mehrerer Bezeichnungen, getrennt durch ein Komma, existieren.

Die Felder, bei denen jetzt kein Eintrag mehr notwendig sind, werden mit einem normalen Standardrahmen versehen. Die erforderliche Eingabe wird auf **`False`** gestellt.

```
                IF InStr(a(n),"notrequired") THEN
                    aa = split(a(n),"[")
                    ab = split(aa(1),",")
                    FOR k = LBound(ab()) TO UBound(ab())
                        oField = oForm.getByName(ab(k))
                        oField.BorderColor = loBorderDefault
                        oField.InputRequired = False
                    NEXT
                END IF
            NEXT
        END IF
    NEXT
END SUB
```

Die folgende Prozedur «NotRequired entspricht in Teilen der vorhergehenden Prozedur. Sie wird allerdings beim Verlassen eines Formularfeldes, nicht beim Wechsel eine Formulars aufgerufen. Hier wird nach dem Verlassen ein anderes Feld auf **Eingabe erforderlich → Nein** gesetzt, wenn das Ausgangsfeld einen Inhalt enthält. Enthält es keinen Inhalt, so muss ist bei **Eingabe erforderlich → Ja** gesetzt. Entsprechend werden auch die Rahmenfarben angepasst.

```
SUB NotRequired(oEvent AS OBJECT)
    DIM oFieldStart AS OBJECT, oForm AS OBJECT
    DIM n AS INTEGER, k AS INTEGER
    DIM a(), aa(), ab()
    oFieldStart = oEvent.Source.Model
    oForm = oFieldStart.Parent
    a = split(oFieldStart.Tag,"]")
    FOR n = LBound(a()) TO UBound(a())-1
        IF InStr(a(n),"notrequired") THEN
            aa = split(a(n),"[")
            ab = split(aa(1),",")
            FOR k = LBound(ab()) TO UBound(ab())
                oField = oForm.getByName(ab(k))
                IF oFieldStart.CurrentValue <> "" THEN
                    oField.BorderColor = loBorderDefault
                    oField.InputRequired = False
                ELSE
                    oField.BorderColor = loBorderInputRequired
                    oField.InputRequired = True
                END IF
            NEXT
        END IF
    NEXT
END SUB
```

Mit der Prozedur «EnableDisable» werden Felder abhängig von einem anderen Feld so eingeschaltet, dass gegebenenfalls keine Eingabe mehr notwendig ist. So steht in den Zusatzinformationen zu dem Markierfeld «Keine Adresse» **`inaktiv[txtStraße,comPLZOrt]`**. Es sollen also die Felder für die «Straße» und für «PLZ – Ort» inaktiv gesetzt werden, wenn das Markierfeld ausgewählt wurde (**`State = True`**)

Der Zugriff ist hier gleich dem der vorhergehenden Prozeduren. Wenn die Eingabe nicht mehr möglich sein soll, dann werden sowohl Rahmen als auch Hintergrundfarbe des Feldes auf die Standardrahmenfarbe eingestellt.

```
SUB EnableDisable(oEvent AS OBJECT)
    DIM oForm AS OBJECT, oField AS OBJECT
    DIM stTag AS STRING
    DIM i AS INTEGER, k AS INTEGER
    DIM a(), aa(), ab()
    oForm = oEvent.Source.Model.Parent
    stTag = oEvent.Source.Model.Tag
    a = split(stTag,"]")
    FOR i = LBound(a()) TO UBound(a())-1
        IF InStr(a(i),"inaktiv") THEN
            aa = split(a(i),"[")
            ab = split(aa(1),",")
            FOR k = LBound(ab()) TO UBound(ab())
                oField = oForm.getByName(ab(k))
                IF oEvent.Source.Model.State THEN
                    oField.Enabled = False
                    oField.BorderColor = loBorderDefault
                    oField.BackgroundColor = loBorderDefault
                ELSE
                    oField.Enabled = True
                    oField.BorderColor = loBorderInputRequired
                    oField.BackgroundColor = loColorStandard
                END IF
            NEXT
        END IF
    NEXT
```

```
END SUB
```

Die Prozedur «FieldRequired» wird an die Felder gebunden, bei denen die Eingabe zu Beginn erforderlich gesetzt wurde. Enthält das Feld keinen Wert, so wird beim Fokusverlust der Rahmen rot dargestellt. Umgekehrt wird der Rahmen auf die Normalfarbe gesetzt, wenn das Feld Inhalt enthält. Ist außerdem in den Zusatzinformationen des Feldes noch das Stichwort 'notrequired' enthalten, dann wird die Prozedur «NotRequired» anschließend gestartet.

```
SUB FieldRequired(oEvent AS OBJECT)
    DIM oField AS OBJECT
    oField = oEvent.Source.Model
    IF oField.CurrentValue <> "" THEN
        oField.BorderColor = loBorderDefault
    ELSE
        oField.BorderColor = loBorderInputRequired
    END IF
    IF inStr(oField.Tag,"notrequired") THEN
        NotRequired(oEvent)
    END IF
END SUB
```

Fehlerhafte Eingaben vermeiden

In der Beispieldatenbank ist für mehrere Felder ein Prozedur eingebaut, die eine fehlerhafte Eingabe so weit wie möglich verhindern soll. Die folgende Prozedur erledigt dies für die Eingabe der IBAN. Sie ist an ein maskiertes Feld gebunden und wird beim Verlassen des Feldes aufgerufen.

```
SUB IBANValid(oEvent AS OBJECT)
    DIM oField AS OBJECT, oForm AS OBJECT, oController AS OBJECT, oView AS OBJECT
    DIM stMsg AS STRING, stText AS STRING, stLand AS STRING, stPruef AS STRING
    DIM i AS INTEGER
    DIM a()
    oField = oEvent.Source.Model
    stText = oField.Text
```

Nur wenn das Feld Text enthält soll die Prozedur auch ablaufen. Das bedeutet, wenn nur einmal der Cursor in dem maskierten Feld gelandet ist und keine Eingabe gemacht wurde ist auch nichts zu überprüfen. Bei der ersten Eingabe, die auch ruhig wieder gelöscht werden kann, würde allerdings die Eingabemaske als Text angesehen. Der Text würde, sofern ein Eintrag fehlt, jetzt mindestens einen Unterstrich '_' enthalten. Außerdem würde der Beginn des Textes 'DE' lauten.

```
    IF stText <> "" THEN
        IF inStr(stText,"_") THEN
            IF Val(Mid(stText,3)) > 0 THEN
                stMsg = "Die IBAN ist zu kurz."
```

Aus dem Text wird ab dem 3. Zeichen versucht, den Wert einer Dezimalzahl auszulesen. Schließlich wird die IBAN ab dem 3. Zeichen nur aus Zahlen zusammengesetzt. Leerzeichen ignoriert die Funktion **`Val()`**. Ist der Wert größer als 0 und enthält der Text gleichzeitig Unterstriche, so ist die IBAN-Angabe zu kurz. Ist der Wert 0, so soll keine Eingabe erfolgt sein. Das Feld wird mit dem Kommando **`reset`** zurückgesetzt und erscheint als leerer Text.

```
            ELSE
                oField.reset
            END IF
```

Enthält das maskierte Feld an jeder Stelle Zeichen, so ist prinzipiell das Format korrekt. In Vierergruppen sind die Zahlen gebündelt eingegeben und vollständig. Die erste Vierergruppe enthält die Landesbezeichnung und die zweistellige Prüfziffer. Die Gruppen werden hier als ein Array aufgetrennt. Trenner ist standardmäßig das Leerzeichen.

```
        ELSE
            a = split(stText)
            stLand = "1314" & Right(a(0),2)
```

Der Landescode wird in Zahlen umgesetzt. 'A': '10', 'B': '11', 'C': '12', 'D': '13', 'E': '14' usw., so dass aus 'DE' die Kombination '1314' wird. Dieser Kombination wird nach die Prüfziffer hinzugefügt. Die Berechnung der Korrektheit der Prüfziffer erfolgt nach dem Prinzip

1. alle Zahlen ab der 3. Zahl zusammen mit der Länderzahl und der Prüfziffer ergeben die Gesamtzahl
2. Die Gesamtzahl wird durch 97 geteilt
3. Aus der Ganzzahldivision muss ein Rest von 1 hervorgehen.

Leider ist dieses Verfahren nicht so einfach möglich, weil die Gesamtzahl zu groß ist. Der Variablentyp LONG kann maximal 2147483648 annehmen, aber nicht 24 Stellen. Das Rechenverfahren wird hier wie eine schriftliche Division in der Schule umgesetzt: Rest berechnen, weitere Werte hinzuholen, Rest berechnen usw. Nur bei der ersten Berechnung können hier 8 Zahlen aus dem Array übernommen werden. Ist dort der Rest über 21, so würde bereits die zweite Teilberechnung fehl schlagen. Deshalb wird bei den folgenden Teilberechnung jeweils nur mit einer Zugabe von einem Arrayelement, das eben 4 Zahlen enthält, weiter gerechnet.

```
            i = cLng(a(1) & a(2)) Mod 97
            i = cLng(i & a(3)) Mod 97
            i = cLng(i & a(4)) Mod 97
            i = cLng(i & a(5)) Mod 97
            i = cLng(i & stLand) Mod 97
            IF i <> 1 THEN
```

Ist die Prüfung fehl geschlagen, weil der Rest nicht gleich '1' ist, so wird hier als kleiner Zusatz noch die eventuell mögliche Prüfziffer berechnet. Dies ist bei einer angenommenen Prüfziffer von '00' der Rest zu '98'. Eine Gewähr für eine korrekte IBAN bietet dies aber nicht, da ja der Fehler auch an anderer Stelle innerhalb der IBAN liegen kann.

Ist die Prüfung fehl geschlagen, so muss eine Fehlermeldung auf dem Bildschirm präsentiert werden.

```
                stPruef = "131400"
                i = cLng(a(1) & a(2)) Mod 97
                i = cLng(i & a(3)) Mod 97
                i = cLng(i & a(4)) Mod 97
                i = cLng(i & a(5)) Mod 97
                i = cLng(i & stPruef) Mod 97
                i = 98 - i
                stMsg "Die IBAN ist fehlerhaft." & CHR(13) & "Die Prüfziffer müsste "
                   & i & " sein."
            END IF
        END IF
```

Erfolgte eine Fehlermeldung, so darf die Prüfung hiermit aber nicht abgeschlossen sein. Der Cursor muss so lange ist das Eingabefeld zurückgesetzt werden bis die Prüfung das Ergebnis annimmt. Jede Prüfroutine sucht also bei einem Fehler anschließend den Controller des Dokumentes auf und setzt den Cursor in das Feld zurück.

```
        IF stMsg <> "" THEN
            msgbox (stMsg, 0, "Eingabe fehlerhaft")
            oForm = oField.Parent
            oController = thisComponent.getCurrentController()
            oView = oController.getControl(oForm.getByname(oField.Name))
            oView.setFocus
        END IF
    END IF
END SUB
```

Abspeichern nach erfolgter Kontrolle

Die Felder, bei denen eine Eingabe notwendig sind, verhindern nicht, dass eine Person dennoch eine Abspeicherung der Daten vornehmen will. Mit der folgenden Funktion «SaveRequired» wird jetzt noch einmal überprüft, ob in allen Feldern, für die eine Eingabe notwendig ist, auch eine Eingabe steht. Ansonsten wird die Speicherung unterbrochen und eine Fehlermeldung ausgegeben.

```
FUNCTION SaveRequired(oEvent AS OBJECT) AS BOOLEAN
    DIM oForm AS OBJECT, oField AS OBJECT
    DIM stLabel AS STRING
    DIM k AS INTEGER, i AS INTEGER
```

```
    oForm = oEvent.Source
    IF oForm.ImplementationName = "org.openoffice.comp.svx.FormController" THEN
```

Beim Abspeichern wird zuerst der «FormController» aktiviert. Anschließend auch noch die Implementation für «ODatabaseForm». Das Auslesen der Felder ist hier unterschiedlich, so dass direkt der «FormController» zur Auswertung genutzt wird. In dem «FormController» sind die einzelnen Felder nur über das Model erreichbar.

```
        FOR i = 0 TO oForm.Model.Count - 1
            oField = oForm.Model.getByIndex(i)
            IF oField.supportsService("com.sun.star.form.DataAwareControlModel") THEN
                IF oField.InputRequired AND NOT IsNULL(oField.BoundField) THEN
                    IF oField.BoundField.String = "" THEN
                        stLabel = stLabel & ", " & oField.LabelControl.Label
                        k = k + 1
                    END IF
                ELSEIF oField.InputRequired THEN
                    IF oField.CurrentValue = "" THEN
                    stLabel = stLabel & ", " & oField.LabelControl.Label
                    k = k + 1
                    END IF
                ELSE
                END IF
            END IF
        NEXT
```

Die Schleife erfolgt hier in zwei Schritten. In dem Formular befindet sich ein Bildfeld, das die Eigenschaft CurrentValue nicht bedienen kann. Es befindet sich aber auch ein Kombinationsfeld darin, das gar nicht an die zugrundeliegende Tabelle gekoppelt ist und damit kein gebundenes Feld der Tabelle ansprechen kann.

Ist der Zähler größer als 1, so muss eine Fehlermeldung erfolgen. Hier wurde bereits über die den Feldern zugewiesenen Beschriftungsfelder (**`Label`**) entsprechend die lesbare Bezeichnung der leeren Felder herausgesucht. «stLabel» endet allerdings mit einem Komma, gefolgt von einer Leertaste. Dies ist für den Abschluss des Strings überflüssig und wird abgetrennt.

```
        IF k > 0 THEN
            stLabel = Right(stLabel, Len(stLabel)-2)
            IF k = 1 Then
                stText = "Für das Feld "
            ELSE
                stText = "Für die Felder "
            END IF
            MsgBox ("Alle rot umrandeten Felder müssen einen Inhalt aufweisen." &
                CHR(13) & stText & stLabel & " ist eine Eingabe erforderlich." ,
                0 ,"Speichern gestoppt")
            SaveRequired = False
            EXIT FUNCTION
        ELSE
            SaveRequired = True
        END IF
    END IF
END FUNCTION
```

Bei einem Fehler gibt die Funktion **`False`** zurück. Die Abspeicherung wird unterbrochen und eine Suche nach den Fehlern kann beginnen.

Primärschlüssel aus Nummerierung und Jahreszahl

Bei der Erstellung von Rechnungen werden jährlich Bilanzen gezogen. Das führt manchmal zu dem Wunsch, die Rechnungstabellen einer Datenbank nach Jahren getrennt zu sichern und jedes Jahr mit einer neuen Tabelle zu beginnen.

Die folgende Makrolösung geht einen anderen Weg. Sie schreibt automatisch den Wert für das Feld «ID» in die Tabelle, berücksichtigt dabei aber das «Jahr», das in der Tabelle als zweiter Pri-

märschlüssel existiert. So tauchen dann in der Tabelle als Primärschlüssel z.B. die folgenden Werte auf:[11]

Jahr	ID
2014	1
2014	2
2014	3
2015	1
2015	2

Damit lässt sich eine auf das Jahr bezogene Übersicht auch in den Dokumenten besser erzeugen.

```
SUB Datum_aktuell_ID_einfuegen
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM stSql AS STRING
    DIM oAbfrageergebnis AS OBJECT
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM oFeld2 AS OBJECT
    DIM oFeld3 AS OBJECT
    DIM inIDneu AS INTEGER
    DIM inYear AS INTEGER
    DIM unoDate
    oDoc = thisComponent
    oDrawpage = oDoc.drawpage
    oForm = oDrawpage.forms.getByName("MainForm")
    oFeld1 = oForm.getByName("fmtJahr")
    oFeld2 = oForm.getByName("fmtID")
    oFeld3 = oForm.getByName("datDatum")
    IF IsEmpty(oFeld2.getCurrentValue()) THEN
        IF IsEmpty(oFeld3.getCurrentValue()) THEN
            unoDate = createUnoStruct("com.sun.star.util.Date")
            unoDate.Year = Year(Date)
            unoDate.Month = Month(Date)
            unoDate.Day = Day(Date)
            inYear = Year(Date)
        ELSE
            inYear = oFeld3.CurrentValue.Year
        END IF
        oDatenquelle = ThisComponent.Parent.CurrentController
        If NOT (oDatenquelle.isConnected()) THEN
            oDatenquelle.connect()
        END IF
        oVerbindung = oDatenquelle.ActiveConnection()
        oSQL_Anweisung = oVerbindung.createStatement()
        stSql = "SELECT MAX( ""ID"" )+1 FROM ""Auftraege"" WHERE ""Jahr"" = '"
            + inYear + "'"
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        WHILE oAbfrageergebnis.next
            inIDneu = oAbfrageergebnis.getInt(1)
        WEND
        IF inIDneu = 0 THEN
            inIDneu = 1
        END IF
        oFeld1.BoundField.updateInt(inYear)
        oFeld2.BoundField.updateInt(inIDneu)
        IF IsEmpty(oFeld3.getCurrentValue()) THEN
```

11 Dem Handbuch liegt die Datenbank «Beispiel_Fortlaufende_Nummer_Jahr.odb» bei.

```
            oFeld3.BoundField.updateDate(unoDate)
        END IF
    END IF
END SUB
```

Alle Variablen werden deklariert. Die Formularfelder in dem Hauptformular werden angesteuert. Der Rest des Codes läuft nur ab, wenn der Eintrag für das Feld «fmtID» noch leer ist. Dann wird, wenn nicht schon ein Datum eingegeben wurde, zuerst ein Datumsstruct erstellt, um das aktuelle Datum und das aktuelle Jahr in die entsprechenden Felder übertragen zu können. Anschließend wird der Kontakt zu der Datenbank aufgebaut, sofern noch kein Kontakt existiert. Es wird zu dem höchsten Eintrag des Feldes "ID", bezogen auf das Jahr des Datumsfeldes, der Wert '1' addiert. Bleibt die Abfrage leer, so existiert noch kein Eintrag in dem Feld "ID". Jetzt könnte genauso gut '0' direkt in das Feld «fmtID» eingetragen werden. Die Nummerierung für die Aufträge sollte aber mit '1' beginnen, so dass der Variablen «inIDneu» eine '1' zugewiesen wird.

Die ermittelten Werte für das Jahr, die ID und, sofern nicht bereits ein Datum eingetragen wurde, das aktuelle Datum, werden schließlich in das Formular übertragen.

Im Formular sind die Felder für die Primärschlüssel "ID" und "Jahr" schreibgeschützt. Die Zuweisung kann so nur durch das Makro erfolgen.

Datenbankaufgaben mit Makros erweitert

Verbindung mit Datenbanken erzeugen

```
oDatenquelle = ThisComponent.Parent.DataSource
IF NOT oDatenquelle.IsPasswordRequired THEN
    oVerbindung = oDatenquelle.GetConnection("","")
```

Hier wäre es möglich, fest einen Benutzernamen und ein Passwort einzugeben, wenn eine Passworteingabe erforderlich wäre. In den Klammer steht dann ("Benutzername","Passwort").

Statt einen Benutzernamen und ein Passwort in Reinschrift einzutragen, wird für diesen Fall der Dialog für den Passwortschutz aufgerufen:

```
ELSE
    oAuthentifizierung = createUnoService("com.sun.star.sdb.InteractionHandler")
    oVerbindung = oDatenquelle.ConnectWithCompletion(oAuthentifizierung)
END IF
```

Dies funktioniert aber nicht, wenn bei der Verbindung bereits eine Benutzername- und Passworteingabe in Base vorgegeben wurde. Hier muss mit

```
oDatenquelle.Password = "mein Passwort"
```

das Passwort der Verbindung mitgegeben werden. Dann erscheint der Dialog nicht mehr. Anschließend muss noch mit der Datenquelle verbunden werden, um direkt auf z.B. ein Formular zugreifen zu können:

```
ThisComponent.CurrentController.Connect()
```

Wird allerdings von einem Formular innerhalb der Base-Datei auf die Datenbank zugegriffen, so reicht bereits

```
oDatenquelle = ThisComponent.Parent.CurrentController
IF NOT (oDatenquelle.isConnected()) Then
    oDatenquelle.connect()
End IF
oVerbindung = oDatenquelle.ActiveConnection()
```

Die Datenbank ist hier bekannt, ein Nutzername und ein Passwort sind nicht erforderlich, da diese bereits in den Grundeinstellungen der HSQLDB für die interne Version ausgeschaltet sind.

Für Formulare außerhalb von Base wird die Verbindung über das erste Formular hergestellt:

```
oDatenquelle = Thiscomponent.Drawpage.Forms(0)
oVerbindung = oDatenquelle.activeConnection
```

Daten von einer Datenbank in eine andere kopieren

Die interne Datenbank ist erst einmal eine Ein-Benutzer-Datenbank. Die Daten werden innerhalb der *.odb-Datei abgespeichert. Ein Austausch von Daten zwischen verschiedenen Datenbankdateien ist eigentlich nicht vorgesehen, über Export und Import allerdings möglich.

Manchmal werden aber auch *.odb-Dateien so eingesetzt, dass ein möglichst automatischer Datenaustausch von einer Datenbankdatei zu einer anderen erfolgen soll. Die folgende Prozedur kann da hilfreich sein.[12]

Nach der Deklaration der Variablen wird der Pfad der aktuellen Datenbankdatei von einem Button im Formular aus ausgelesen. Von dem Pfad wird der Dateiname abgetrennt. Die Zieldatei für die Daten befindet sich ebenfalls in dem Verzeichnis. Der Name dieser Datei wird jetzt an den Pfad angehängt, damit der Kontakt zur Zieldatenbankdatei erstellt werden kann.

Der Kontakt zur Ausgangsdatenbank wird im Verhältnis zum Formular ermittelt, in dem der Button liegt: **`ThisComponent.Parent.CurrentController`**. Der Kontakt zur externen Datenbank wird über den **`DatabaseContext`** und den Pfad zur Datenbank erstellt.

```
SUB Datenkopie
    DIM oDatabaseContext AS OBJECT
    DIM oDatenquelle AS OBJECT
    DIM oDatenquelleZiel AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oVerbindungZiel AS OBJECT
    DIM oDB AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM oSQL_AnweisungZiel AS OBJECT
    DIM oAbfrageergebnis AS OBJECT
    DIM oAbfrageergebnisZiel AS OBJECT
    DIM stSql AS String
    DIM stSqlZiel AS String
    DIM inID AS INTEGER
    DIM inIDZiel AS INTEGER
    DIM stName AS STRING
    DIM stOrt AS STRING
    oDB = ThisComponent.Parent
    stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
    stDir = ConvertToUrl(stDir & "ZielDB.odb")
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
       oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
    oDatenquelleZiel = oDatabaseContext.getByName(stDir)
    oVerbindungZiel = oDatenquelleZiel.GetConnection("","")
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "SELECT * FROM ""Tabelle"""
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
    WHILE oAbfrageergebnis.next
       inID = oAbfrageergebnis.getInt(1)
       stName = oAbfrageergebnis.getString(2)
       stOrt = oAbfrageergebnis.getString(3)
       oSQL_AnweisungZiel = oVerbindungZiel.createStatement()
       stSqlZiel = "SELECT ""ID"" FROM ""Tabelle"" WHERE ""ID"" = '"+inID+"'"
       oAbfrageergebnisZiel = oSQL_AnweisungZiel.executeQuery(stSqlZiel)
       inIDZiel = - 1
       WHILE oAbfrageergebnisZiel.next
          inIDZiel = oAbfrageergebnisZiel.getInt(1)
```

12 Das Beispiel "Datenkopie_Quelle_Ziel" ist als gepacktes Verzeichnis diesem Handbuch beigefügt.

```
        WEND
        IF inIDZiel = - 1 THEN
            stSqlZiel = "INSERT INTO ""Tabelle"" (""ID"",""Name"",""Ort"") VALUES
                ('"+inID+"','"+stName+"','"+stOrt+"')"
            oSQL_AnweisungZiel.executeUpdate(stSqlZiel)
        END IF
    WEND
END SUB
```

Die komplette Tabelle der Ausgangsdatenbank wird ausgelesen und Zeile für Zeile anschließend über den Kontakt zur Zieldatenbank in die Tabelle der Zieldatenbank eingefügt. Vor dem Einfügen wird allerdings getestet, ob der Wert für den Primärschlüssel bereits vorhanden ist. Ist der Schlüsselwert vorhanden, so wird der Datensatz nicht kopiert.

Hier könnte gegebenenfalls auch eingestellt werden, dass statt einer Kopie des Datensatzes ein Update des bereits existierenden Datensatzes erfolgen soll. Auf jeden Fall wird so sichergestellt, dass die Zieldatenbank die Datensätze mit den entsprechenden Primärschlüsseln der Quelldatenbank enthält.

Direkter Import von Daten aus Calc

Häufig passiert es, dass Calc statt einer Datenbank zur Eingabe von Daten in eine Tabelle genutzt wird. Solche Daten lassen sich dann über die Zwischenablage oder per Drag-And-Drop in eine Base-Tabelle einlesen. Auch der Export in eine bereits in Base eingebundene *.csv-Textdatei ist möglich.

Soll allerdings Calc auf Dauer zur Dateneingabe genutzt werden und die Daten regelmäßig aus Calc ausgelesen werden, so ist der Kopierschritt vielleicht zu umständlich. Hier setzt das folgende Makro an, das aus einem Formular heraus über einen Button gestartet wird.[13]

Das Makro geht von folgenden Voraussetzungen aus:

1. Die Daten liegen auf dem ersten Tabellenblatt des Calc-Dokuments
2. Auf diesem Tabellenblatt liegen nur die Daten in einer Spalte, nicht zusätzliche Einträge.
3. Die erste Datenzeile enthält die Feldbenennungen, die genau den Feldbezeichnungen in der Base-Tabelle entsprechen.

Die Dateneingabe muss nicht links oben auf dem Tabellenblatt erfolgen. Auch müssen die Felder nicht die gleiche Reihenfolge wie in der Tabelle haben. Spalten, deren Spaltenüberschrift nicht Feldern der Tabelle entspricht, werden ignoriert.

```
Sub CalcDataImport(oEvent AS OBJECT)
    DIM oDatasource AS OBJECT
    DIM oConnection AS OBJECT
    DIM oSQL_Command AS OBJECT
    DIM oResult AS OBJECT
    DIM oDB AS OBJECT
    DIM oDoc AS OBJECT
    DIM oDocView AS OBJECT
    DIM oRange AS OBJECT
    DIM Arg()
    DIM aColumn()
    DIM aColumns()
    DIM aType()
    DIM stSql AS STRING
    DIM stRow AS STRING
    DIM stDir AS STRING
    DIM stColumns AS STRING
    DIM stStartCol AS STRING
    DIM stEndCol AS STRING
    DIM stStartRow AS STRING
    DIM stEndRow AS STRING
    DIM stRange AS STRING
```

13 Die Beispieldatenbank «Beispiel_Daten_Import.odb» liegt diesem Handbuch bei.

```
DIM inCounter AS INTEGER
DIM loColumns AS LONG
DIM loRows AS LONG
DIM loID AS LONG
oDatasource = ThisComponent.Parent.CurrentController
If NOT (oDatasource.isConnected()) THEN
   oDatasource.connect()
END IF
oConnection = oDatasource.ActiveConnection()
oSQL_Command = oConnection.createStatement()
```

Nach der Deklaration der Variablen werden Spaltennamen und Spaltentypen aus der vorgegebenen Tabelle der Datenbank ausgelesen. Der Primärschlüssel der Tabelle mit der Bezeichnung "ID" wird als Integer-Feld unabhängig von der Calc-Tabelle erstellt.

```
stSql = "SELECT COLUMN_NAME, TYPE_NAME FROM INFORMATION_SCHEMA.SYSTEM_COLUMNS
   WHERE TABLE_NAME = 'Tabelle' AND NOT COLUMN_NAME = 'ID'"
oResult = oSQL_Command.executeQuery(stSql)
inCounter = 0
stColumns = ""
```

Die Spaltennamen und Spaltentypen werden ausgelesen und in getrennten Arrays gespeichert.

Hinweis

Für Firebird muss der SQL-Code angepasst werden:

```
stSql = "SELECT A.RDB$FIELD_NAME, C.RDB$TYPE_NAME
   FROM RDB$RELATION_FIELDS AS A, RDB$FIELDS AS B, RDB$TYPES AS C
   WHERE A.RDB$FIELD_SOURCE = B.RDB$FIELD_NAME
      AND B.RDB$FIELD_TYPE = C.RDB$TYPE
      AND C.RDB$FIELD_NAME = 'RDB$FIELD_TYPE'
      AND A.RDB$RELATION_NAME = 'Tabelle'
      AND NOT A.RDB$FIELD_NAME = 'ID'"
```

```
WHILE oResult.next
   ReDim Preserve aColumn(inCounter)
   ReDim Preserve aType(inCounter)
   aColumn(inCounter) = oResult.getString(1)
   aType(inCounter) = oResult.getString(2)
   inCounter = inCounter+1
WEND
```

Der zur Zeit höchste Eintrag für den Primärschlüsselwert wird ermittelt und um 1 erhöht. In diesem Beispiel wird also der Schlüsselwert nicht automatisch von der HSQLDB hoch geschrieben, sondern über das Makro verwaltet. Dies geschieht hier zu Testzwecken, da die Tabelle laufend unter unterschiedlichen Kriterien testweise eingelesen wurde. Entsprechend könnte natürlich auch der Tabellenindex heruntergesetzt werden, wie dies in dem Makro «Tabellenindex_runter» passiert.

```
stSql = "SELECT MAX(""ID"") FROM ""Tabelle"""
oResult = oSQL_Command.executeQuery(stSql)
WHILE oResult.next
   loID = oResult.getInt(1) + 1
WEND
```

Der Pfad zur Calc-Datei wird anhand der Lage der Base-Datei im Dateisystem ermittelt. Die Calc-Datei liegt hier im gleichen Verzeichnis wie die Base-Datei.

Anschließend wird die Calc-Datei geladen und gleich unsichtbar geschaltet, damit sie sich nicht in den Vordergrund schiebt.

```
oDB = ThisComponent.Parent
stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
stDir = ConvertToUrl(stDir & "Daten_Calc.ods")
oDoc = StarDesktop.loadComponentFromURL(stDir,"_blank", 0, Arg() )
oDocView = oDoc.CurrentController.Frame.ContainerWindow
oDocView.Visible = False
```

Die Position des beschrifteten Bereiches des Tabellenblattes wird ermittelt. Dies geschieht über die **`ColumnDescriptions`** und **`RowDescriptions`**. Sie geben genau die Anzahl der beschrifteten

Spalten und Zeilen wieder. Außerdem kann darüber die Bezeichnung der Spalte und der Zeile wie z.B. «B2» ausgelesen werden.

```
loColumns = uBound(oDoc.Sheets(0).ColumnDescriptions)' Anzahl der Spalten
stStartCol = split(oDoc.Sheets(0).ColumnDescriptions(0))(1)
stEndCol = split(oDoc.Sheets(0).ColumnDescriptions(loColumns))(1)
loRows = uBound(oDoc.Sheets(0).RowDescriptions)   ' Anzahl der Zeilen
stStartRow = split(oDoc.Sheets(0).RowDescriptions(0))(1)
stEndRow = split(oDoc.Sheets(0).RowDescriptions(loRows))(1)
stRange = stStartCol & stStartRow & ":" & stEndCol & StEndRow
```

Der Bereich wird als String zusammengesetzt. Dies erfolgt in der gleichen Art und Weise wie bei der Benennung innerhalb von Calc, also z.B. «A1:C7». Die Daten aus so einem Bereich können mit der Funktion **`GetDataArray()`** ausgelesen werden.

```
oRange = oDoc.Sheets(0).getCellRangeByName(stRange)
aDat = oRange.getDataArray()
```

Die Spaltennamen stehen in der ersten Zeile. Sie können eine andere Reihenfolge haben, als dies innerhalb der Tabelle von Base vorgesehen wird. Deshalb werden diese Bezeichnungen für einen späteren Vergleich in einem separaten Array gespeichert. Sie werden in der folgenden Schleife nicht als Werte zum Einlesen in die Tabelle abgefragt. Deshalb beginnt die Schleife mit **`i=1`**.

```
aColumns = aDat(0)
FOR i = 1 TO uBound(aDat)
    aRow = aDat(i)
    stColumns = """ID"""
    stRow = loID
    FOR k = 0 TO loColumns
        FOR n = 0 TO uBound(aColumn)
```

Im folgenden werden die Spaltenbezeichnungen aus Calc mit denen der Base-Tabelle verglichen. Die Base-Tabelle enthält hier neben Textspalten auch eine Spalte für einen Währungsbetrag, die als **`DECIMAL`** definiert ist, sowie ein Datum.

Bei dem Währungsbetrag muss das Dezimalkomma gegebenenfalls zu einem Dezimalpunkt umgewandelt werden. Deshalb hier die Funktion **`Cdbl`**, gekoppelt mit der Funktion **`Str`**.

Bei dem Datumsfeld gibt Calc den Inhalt als ISO-Zahlencode aus. Dieser Code muss zuerst daraufhin überprüft werden, ob denn überhaupt ein Datum daraus gebildet werden kann. Ist dies möglich, so wird ein SQL-konformes Datum im Format YYYY-MM-DD zusammengestellt, das auch bei einstelligen Tageswerten und Monatswerten nicht versagt.

```
IF aColumns(k) = aColumn(n) THEN
    IF aType(n) = "DECIMAL" THEN
        IF aRow(k) <> "" THEN
            aRow(k) = Str(CDbl(aRow(k)))
        END IF
    END IF
    IF aType(n) = "DATE" THEN
        IF isDate(CDate(aRow(k))) AND aRow(k) <> "" THEN
            aRow(k) = Year(CDate(aRow(k)))&"-"
               &Right("0"&Month(CDate(aRow(k))),2)
               &"-"&Right("0"&Day(CDate(aRow(k))),2)
        ELSE
            aRow(k) = ""
        END IF
    END IF
```

Ist der Zellinhalt leer oder für ein Dezimalfeld bzw. Datumsfeld gegebenenfalls nicht gültig, so wird an die Base-Tabelle **`NULL`** weitergegeben. Andernfalls wird der Inhalt in Hochkommata gesetzt. Anschließend werden die Inhalte über Kommata miteinander verbunden. Auch die Bezeichnung der Spalten erfolgt in der Reihenfolge, in der sie aus der Calc-Tabelle ausgelesen wurden.

```
    IF aRow(k) = "" THEN
        aRow(k) = "NULL"
    ELSE
        aRow(k) = "'" & aRow(k) & "'"
```

```
                END IF
                stRow = stRow & "," & aRow(k)
                stColumns = stColumns & ",""" & aColumns(k) & """"
            END IF
        NEXT
    NEXT
    stSql = "INSERT INTO ""Tabelle"" ("& stColumns &") VALUES ("& stRow &")"
    oSQL_Command.executeUpdate(stSql)
```

Der Primärschlüsselwert wird hier innerhalb des Makros um 1 heraufgesetzt.

```
        loID = loID + 1
    NEXT
    oDoc.close(True) ' Schließen des Calc-Documentes
    oEvent.Source.Model.parent.reload()  ' Neuladen des Formulars
End Sub
```

Ist der gesamte Inhalt aus dem Calc-Tabellenblatt ausgelesen, so wird das unsichtbare Dokument geschlossen. Anschließend wird das Formular, in dem sich der auslösende Button befindet, neu eingelesen. Dabei wird das Formular über den Button mit **`oEvent.Source.Model.parent`** ermittelt.

Zugriff auf Abfragen

Abfragen lassen sich in der grafischen Benutzeroberfläche einfacher zusammenstellen als den gesamten Text in Makros zu übertragen, zumal dann auch noch innerhalb des Makros alle Felder- und Tabellenbezeichnungen in zweifach doppelte Anführungszeichen gesetzt werden müssen.

```
SUB Abfrageninhalt
    DIM oDatenDatei AS OBJECT
    DIM oAbfragen AS OBJECT
    DIM stQuery AS STRING
    oDatenDatei = ThisComponent.Parent.CurrentController.DataSource
    oAbfragen = oDatenDatei.getQueryDefinitions()
    stQuery = oAbfragen.getByName("Query").Command
    msgbox stQuery
END SUB
```

Aus einem Formular heraus wird auf den Inhalt der *.odb-Datei zugegriffen. Die Abfragen werden über **`getQueryDefinitions`**() ermittelt. Die SQL-Formulierung der Abfrage "Query" wird über den Zusatz **`Command`** bereit gestellt. **`Command`** kann schließlich dazu genutzt werden, eine entsprechende Abfrage auch innerhalb eines Makros weiter zu nutzen.

Allerdings muss bei der Nutzung des SQLCodes der Abfrage darauf geachtet werden, dass sich der Code nicht wiederum auf eine Abfrage bezieht. Das führt dann unweigerlich zu der Meldung, dass die (angebliche) Tabelle der Datenbank unbekannt ist. Einfacher ist es daher, aus Abfragen Ansichten zu erstellen und entsprechend auf die Ansichten in Makros zuzugreifen.

Datenbanksicherungen erstellen

Vor allem beim Erstellen von Datenbanken kann es hin und wieder vorkommen, dass die *.odb-Datei unvermittelt beendet wird. Vor allem beim Berichtsmodul ist ein häufigeres Abspeichern nach dem Editieren sinnvoll.

Ist die Datenbank erst einmal im Betrieb, so kann sie durch Betriebssystemabstürze beschädigt werden, wenn der Absturz gerade während des Schließens der Base-Datei erfolgt. Schließlich wird in diesem Moment der Inhalt der Datenbank in die Datei zurückgeschrieben.

Außerdem gibt es die üblichen Verdächtigen für plötzlich nicht mehr zu öffnende Dateien wie Festplattenfehler usw. Da kann es dann nicht schaden, eine Sicherheitskopie möglichst mit dem aktuellsten Datenstand parat zu haben. Der Datenbestand ändert sich allerdings nicht, während die *.odb-Datei geöffnet ist. Deshalb kann die Sicherung direkt mit dem Öffnen der Datei verbunden werden. Es werden einfach Kopien der Datei in das unter **Extras → Optionen → LibreOffice → Pfade**

angegebene Backup-Verzeichnis erstellt. Dabei beginnt das Makro nach einer voreingestellten Zahl an Kopien (**inMax**) damit, die jeweils älteste Variante zu überschreiben.

```
SUB Datenbankbackup(inMax AS INTEGER)
    DIM oPath AS OBJECT
    DIM oDoc AS OBJECT
    DIM sTitel AS STRING
    DIM sUrl_Ziel AS STRING
    DIM sUrl_Start AS STRING
    DIM i AS INTEGER
    DIM k AS INTEGER
    oDoc = ThisComponent
    sTitel = oDoc.Title
    sUrl_Start = oDoc.URL
    DO WHILE sUrl_Start = ""
        oDoc = oDoc.Parent
        sTitel = oDoc.Title
        sUrl_Start = oDoc.URL
    LOOP
```

Wird das Makro direkt beim Start der *.odb-Datei ausgeführt, so stimmen **sTitel** und **sUrl_Start**. Wird es hingegen von einem Formular aus ausgeführt, so muss erst einmal ermittelt werden, ob überhaupt eine URL verfügbar ist. Ist die URL leer, so wird eine Ebene höher (**oDoc.Parent**) nach einem Wert nachgesehen.

```
    oPath = createUnoService("com.sun.star.util.PathSettings")
    FOR i = 1 TO inMax + 1
        IF NOT FileExists(oPath.Backup & "/" & i & "_" & sTitel) THEN
            IF i > inMax THEN
                FOR k = inMax - 1 TO 1 STEP -1
                IF FileDateTime(oPath.Backup & "/" & k & "_" & sTitel) <=
                    FileDateTime(oPath.Backup & "/" & k+1 & "_" & sTitel) THEN
                    IF k = 1 THEN
                            i = k
                            EXIT FOR
                    END IF
                ELSE
                    i = k + 1
                    EXIT FOR
                END IF
                NEXT
            END IF
            EXIT FOR
        END IF
    NEXT
    sUrl_Ziel = oPath.Backup & "/" & i & "_" & sTitel
    FileCopy(sUrl_Start,sUrl_Ziel)
END SUB
```

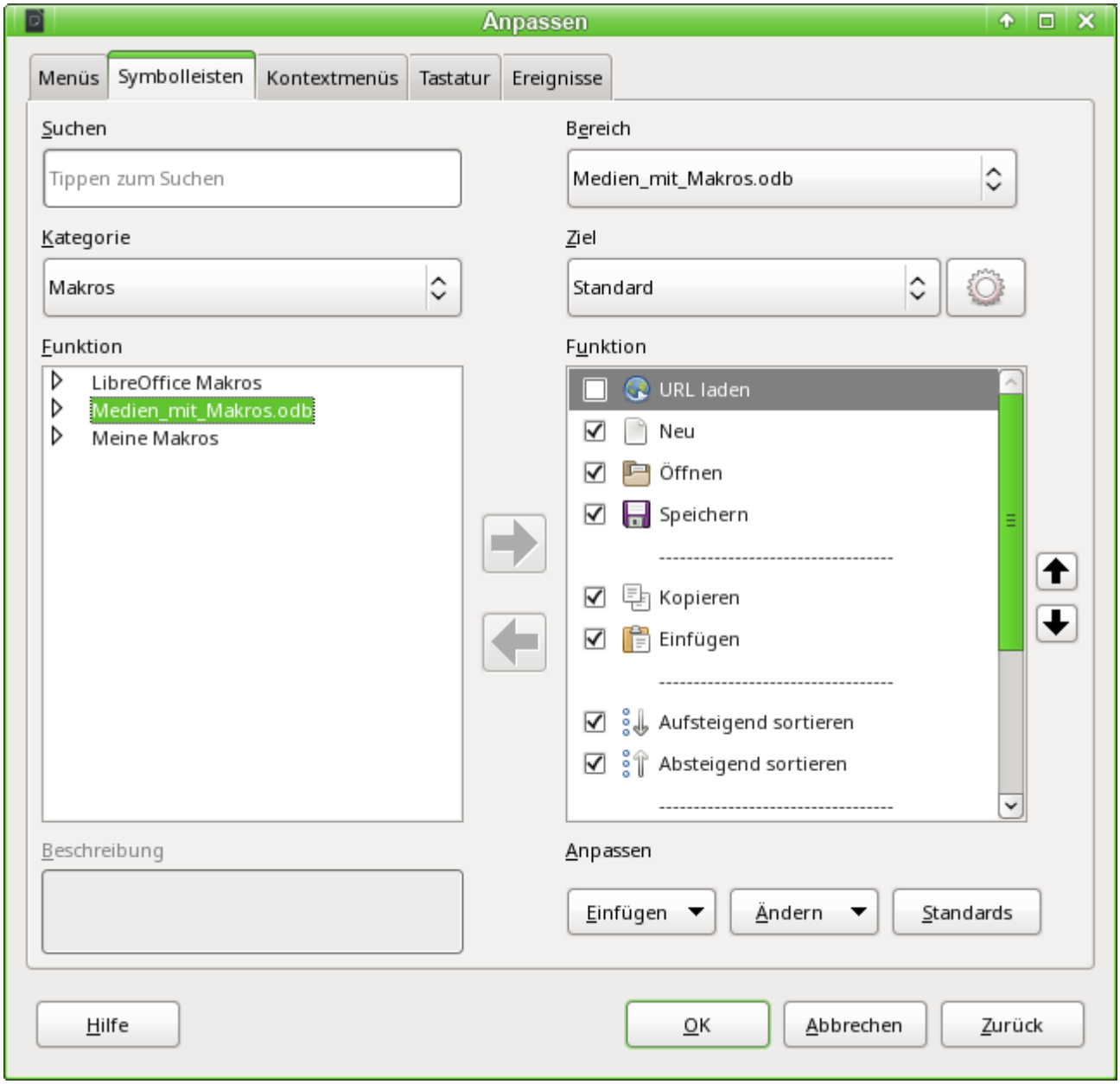
Werden vor der Ausführung der Prozedur «Datenbankbackup» während der Nutzung von Base die Daten aus dem Cache in die Datei zurückgeschrieben, so kann ein entsprechendes Backup auch z.B. nach einer bestimmten Nutzerzeit oder durch Betätigung eines Buttons sinnvoll sein. Das Zurückschreiben regelt die folgende Prozedur:

```
SUB Daten_aus_Cache_schreiben
    DIM oDaten AS OBJECT
    DIM oDataSource AS OBJECT
    oDaten = ThisDatabaseDocument.CurrentController
    IF NOT ( oDaten.isConnected() ) THEN oDaten.connect()
    oDataSource = oDaten.DataSource
    oDataSource.flush
END SUB
```

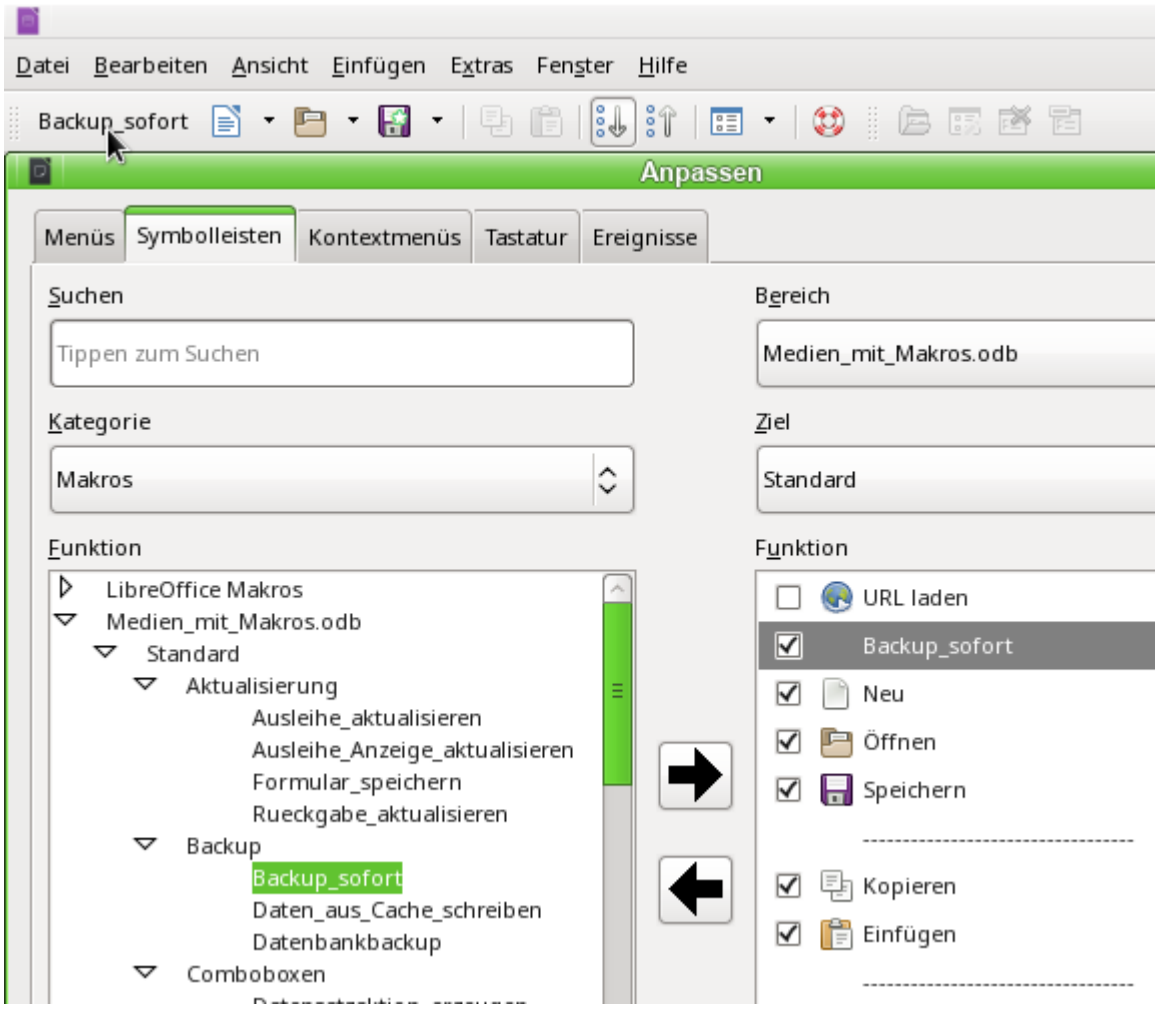
Soll alles zusammen aus einem Formular heraus über einen Button gestartet werden, so müssen beide Prozeduren über eine weitere Prozedur angesprochen werden:

```
SUB Backup_sofort
    Daten_aus_Cache_schreiben
    Datenbankbackup(10)
END SUB
```

Gerade bei einem Sicherungsmakro ist es vielleicht sinnvoll, das Makro über die Symbolleiste der Datenbank erreichbar zu machen. Dies geschieht im Hauptfenster der Base-Datei unter **Extras → Anpassen → Symbolleisten**. Dort wird als **Bereich** die aktuelle Datenbankdatei, als **Kategorie** «Makros» und als **Ziel** die für alle Bereiche zuständige Symbolleiste «Standard» ausgesucht.



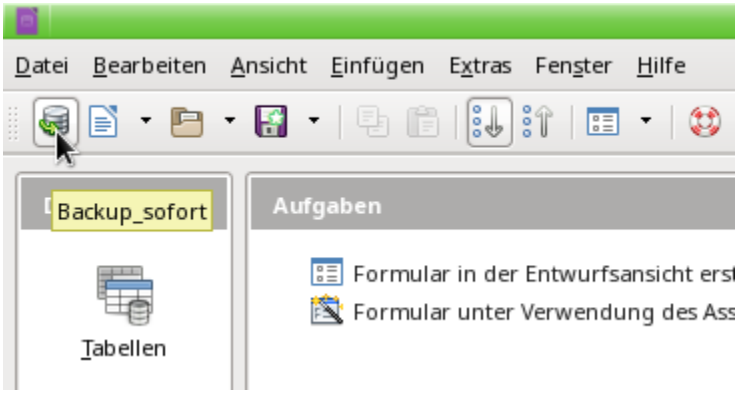
Bei den Makros sind die verfügbaren allgemeinen Makros sowie die Makros aus der Datenbankdatei auswählbar. Aus den Datenbankmakros wird die Prozedur «Backup_Sofort» ausgesucht.



Der Befehl ist jetzt in der Symbolleiste an erster Stelle verfügbar. Um jetzt die Prozedur auszuführen genügt ein Auslösen des Buttons in der Symbolleiste.

Jetzt bietet es sich noch an, dem Befehl ein Symbol zuzuweisen. Über **Ändern → Symbol austauschen** wird der folgende Dialog geöffnet.

Hier wird jetzt ein passendes Symbol gesucht. Es kann auch ein eigenes Symbol erstellt und eingebunden werden.



Das Symbol erscheint anschließend statt der Benennung der Prozedur. Die Benennung wird als Tooltip angezeigt.

Datenbanken komprimieren

Eigentlich nur ein SQL-Befehl (**`SHUTDOWN COMPACT`**), der hin- und wieder, vor allem nach der Löschung von vielen Daten, durchgeführt werden sollte. Die Datenbank speichert neue Daten ab, hält aber gleichzeitig den Platz für die eventuell gelöschten Daten vor. Bei starker Änderung des Datenbestandes muss deshalb der Datenbestand wieder komprimiert werden.

Hinweis	Für die interne HSQLDB wird dieser Befehl seit der LibreOffice-Version 3.6.* automatisch beim Schließen der Datenbank durchgeführt. Diese Makro ist dort also nicht mehr nötig. Bei der ~~Firebird~~-Datenbank gibt es keinen entsprechenden Befehl.

Wird die Komprimierung durchgeführt, so kann anschließend nicht mehr auf die Tabellen zugegriffen werden. Die Datei muss neu geöffnet werden. Deshalb schließt das Makro auch das Formular, aus dem heraus es aufgerufen wird. Leider lässt sich das Dokument selbst nicht schließen, ohne dass beim Neuaufruf die Wiederherstellung anspringt. Deshalb ist diese Funktion auskommentiert.

```
SUB Datenbank_komprimieren
    DIM stMessage AS STRING
    oDatenquelle = ThisComponent.Parent.CurrentController ' Zugriffsmöglichkeit aus dem
        Formular heraus
    IF NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "SHUTDOWN COMPACT" ' Die Datenbank wird komprimiert und geschlossen
    oSQL_Anweisung.executeQuery(stSql)
    stMessage = "Die Datenbank wurde komprimiert." + CHR(13)
        + "Das Formular wird jetzt geschlossen."
    stMessage = stMessage + CHR(13)
        + "Anschließend sollte die Datenbankdatei geschlossen werden."
    stMessage = stMessage + CHR(13) + "Auf die Datenbank kann erst nach dem erneuten
        Öffnen der Datenbankdatei zugegriffen werden."
    msgbox stMessage
    ThisDatabaseDocument.FormDocuments.getByName( "Wartung" ).close
    REM Das Schließen der Datenbankdatei führt beim Neustart zu einem
        Wiederherstellungsablauf.
'   ThisDatabaseDocument.close(True)
END SUB
```

Tabellenindex heruntersetzen bei Autowert-Feldern

Werden viele Daten aus Tabellen gelöscht, so stören sich Nutzer häufig daran, dass die automatisch erstellten Primärschlüssel einfach weiter hochgezählt werden, statt direkt an den bisher höchsten Schlüsselwert anzuschließen. Die folgende Prozedur liest für eine Tabelle den bisherigen Höchstwert des Feldes "ID" aus und stellt den nächsten Schlüsselstartwert um 1 höher als das Maximum ein.

Heißt das Primärschlüsselfeld nicht "ID", so müsste das Makro entsprechend angepasst werden.

```
SUB Tabellenindex_runter(stTabelle AS STRING)
    REM Mit dieser Prozedur wird das automatisch hochgeschriebene Primärschlüsselfeld
        mit der vorgegebenen Bezeichnung "ID" auf den niedrigst möglichen Wert
        eingestellt.
    DIM inAnzahl AS INTEGER
    DIM inSequence_Value AS INTEGER
    oDatenquelle = ThisComponent.Parent.CurrentController' Zugriffsmöglichkeit aus dem
        Formular heraus
    IF NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "SELECT MAX(""ID"") FROM """+stTabelle+"""  ' Der höchste in "ID"
        eingetragene Wert wird ermittelt
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)' Abfrage starten und den
        Rückgabewert in einer Variablen oAbfrageergebnis speichern
    WHILE oAbfrageergebnis.next
        inAnzahl = oAbfrageergebnis.getInt(1)   ' Erstes Datenfeld wird ausgelesen
    WEND   ' nächster Datensatz, in diesem Fall nicht mehr erforderlich, da nur ein
        Datensatz existiert
    IF inAnzahl = "" THEN  ' Falls der höchste Wert gar kein Wert ist, also die
        Tabelle leer ist wird der höchste Wert als -1 angenommen
        inAnzahl = -1
    END IF
    inSequence_Value = inAnzahl+1 ' Der höchste Wert wird um 1 erhöht
    REM Ein neuer Befehl an die Datenbank wird vorbereitet. Die ID wird als neu
        startend ab inAnzahl+1 deklariert.
    REM Diese Anweisung hat keinen Rückgabewert, da ja kein Datensatz ausgelesen
        werden muss
    oSQL_Anweisung1 = oVerbindung.createStatement()
    oSQL_Anweisung1.executeQuery("ALTER TABLE """ + stTabelle + """
        ALTER COLUMN ""ID"" RESTART WITH " + inSequence_Value + "")
END SUB
```

Drucken aus Base heraus

Der Standard, um aus Base heraus ein druckbares Dokument zu erzeugen, ist die Nutzung eines Berichtes. Daneben gibt es noch Möglichkeiten, Tabellen und Abfragen einfach nach Calc zu kopieren und dort zum Druck aufzubereiten. Auch der direkte Druck eines Formularinhaltes vom Bildschirm ist natürlich möglich.

Druck von Berichten aus einem internen Formular heraus

Um einen Bericht zu starten, muss normalerweise die Benutzeroberfläche von Base aufgesucht werden. Ein Mausklick auf den Berichtsnamen startet dann die Ausführung des Berichtes. Einfacher geht es natürlich, den Bericht direkt aus dem Formular heraus zu starten:

```
SUB Berichtsstart
    ThisDatabaseDocument.ReportDocuments.getByName("Bericht").open
END SUB
```

Sämtliche Berichte werden über **`ReportDocuments`** mit ihrem Namen angesprochen. Mit **`open`** werden sie geöffnet. Wird ein Bericht jetzt an eine Abfrage gebunden, die über das Formular gefil-

tert wird, so kann auf diese Art und Weise der zum aktuellen Datensatz anfallende Ausdruck erfolgen.

Start, Formatierung, direkter Druck und Schließen des Berichts

Noch schöner ist es, wenn der Bericht direkt an den Drucker geschickt wird. Die folgende Kombination an Prozeduren legt sogar noch ein paar kleine Features zu. Sie selektiert zuerst den aktiven Datensatz des Formulars, formatiert danach den Bericht um, indem die Felder für den Text auf automatische Höhe eingestellt werden, um anschließend den Bericht zu starten. Schließlich wird der Bericht auch noch gedruckt und ggf. noch als *pdf-Dokument abgespeichert. Und all das passiert nahezu vollständig im Hintergrund, da der Bericht direkt nach dem Öffnen auf unsichtbar geschaltet und nach dem Ausdruck wieder geschlossen wird. Anregungen für die verschiedenen Prozeduren stammen hier von Andrew Piontak, Thomas Krumbein und Lionel Elie Mamane.

```
SUB BerichtStart(oEvent AS OBJECT)
    DIM oForm AS OBJECT
    DIM stSql AS STRING
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM oReport AS OBJECT
    DIM oReportView AS OBJECT
    oForm = oEvent.Source.model.parent
    stSql = "UPDATE ""Filter"" SET ""Integer"" = '" +
        oForm.getInt(oForm.findColumn("ID")) + "' WHERE ""ID"" = TRUE"
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    oSQL_Anweisung.executeUpdate(stSql)
    oReport = ThisDatabaseDocument.ReportDocuments.getByName("Berichtsname").open
    oReportView = oReport.CurrentController.Frame.ContainerWindow
    oReportView.Visible = False
    BerichtZeilenhoeheAuto(oReport)
END SUB
```

Die Prozedur «BerichtStart» wird mit einem Button innerhalb eines Formulars verknüpft. Es kann dabei über den Button der Primärschlüssel des aktuellen Datensatzes des Formulars ausgelesen werden. Dies geschieht hier über das auslösende Ereignis, von dem heraus auf das Formular **`oForm`** geschlossen wird. Der Name des Schlüsselfeldes ist hier mit **`"ID"`** angegeben. Über **`oForm.getInt(oForm.findColumn("ID"))`** wird aus dem Feld der Schlüsselwert als Integer-Wert gelesen. Dieser Wert wird anschließend in einer Filtertabelle abgespeichert. Diese Filtertabelle steuert über eine Abfrage, dass nur der aktuelle Datensatz des Formulars für den Bericht verwendet wird.

Ohne Bezug auf das Formular könnte auch nur der Bericht aufgerufen werden. Dabei ist der aufgerufene Bericht gleich als Objekt ansprechbar (**`oReport`**). Anschließend wird das Fenster auf unsichtbar eingestellt. Dies geht leider nicht direkt mit dem Aufruf, so dass ganz kurz das Fenster erscheint, dann aber ggf. in Ruhe im Hintergrund mit dem entsprechenden Inhalt gefüllt wird.

Anschließend wird die Prozedur «BerichtZeilenhoeheAuto» gestartet. Dieser Prozedur wird der Hinweis auf den geöffneten Bericht mitgegeben.

Die Zeilenhöhe kann zur Zeit nicht beim Berichtsentwurf automatisch angepasst werden. Ist zu viel Text für ein Feld vorgesehen, so wird der Text abgeschnitten und darauf mit Hilfe eines roten Dreiecks hingewiesen. Solange dies nicht funktioniert, stellt die folgende Prozedur sicher, dass z.B. in allen Tabellen mit der Bezeichnung «Detail» die automatische Höhe eingeschaltet wird.

```
SUB BerichtZeilenhoeheAuto(oReport AS OBJECT)
    DIM oTables AS OBJECT
    DIM oTable AS OBJECT
```

```
    DIM inT AS INTEGER
    DIM inI AS INTEGER
    DIM oRows AS OBJECT
    DIM oRow AS OBJECT
    oTables = oReport.getTextTables()
    FOR inT = 0 TO oTables.count() - 1
        oTable = oTables.getByIndex(inT)
        IF Left$(oTable.name, 6) = "Detail" THEN
            oRows = oTable.Rows
            FOR inI = 0 TO oRows.count - 1
                oRow = oRows.getByIndex(inI)
                oRow.IsAutoHeight = True
            NEXT inI
        ENDIF
    NEXT inT
    BerichtDruckenUndSchliessen(oReport)
END SUB
```

Bei dem Entwurf des Berichtes muss darauf geachtet werden, dass alle in einer Zeile des Bereiches «Detail» befindlichen Felder tatsächlich die gleiche Höhe haben. Sonst kann es zusammen mit der Automatik passieren, dass ein Feld plötzlich auf die doppelte Zeilenhöhe gesetzt wird.

Nachdem in allen Tabellen mit der Bezeichnung «Detail» die automatische Höhe eingestellt wurde, wird anschließend der Bericht über die Prozedur «BerichtDruckenUndSchliessen» weiter an den Drucker geschickt.

Das Array Props enthält die verschiedenen Werte, die mit dem Drucker bei einem Dokument verbunden sind. Für den Druckbefehl ist hier lediglich der Name des Standarddruckers wichtig. Das Berichtsdokument soll so lange geöffnet bleiben, bis der Druck tatsächlich abgeschlossen ist. Dies geschieht, indem dem Druckbefehl der Name und der Befehl «Warte, bis ich fertig bin» (**`Wait`**) mitgegeben wird.

```
Sub BerichtDruckenUndSchliessen(oReport AS OBJECT)
    DIM Props
    DIM stDrucker AS STRING
    Props = oReport.getPrinter()
    stDrucker = Props(0).value
    DIM arg(1) AS NEW com.sun.star.beans.PropertyValue
    arg(0).name = "Name"
    arg(0).value = "<" & stDrucker & ">"
    arg(1).name = "Wait"
    arg(1).value = True
    oReport.print(arg())
    oReport.close(true)
End Sub
```

Erst wenn der Druck komplett an den Drucker abgeschickt wurde, wird das Dokument geschlossen.

Zu Einstellungen des Druckers siehe die *Drucker und Druckeinstellungen* aus dem AOO-Wiki.

Soll statt des Drucks oder zusätzlich zu dem Druck auch eine *.pdf-Datei des Dokumentes als Sicherungskopie abgelegt werden, so wird darauf mit der Methode **`storeToURL()`** zugegriffen:

```
Sub BerichtAlsPDFspeichern(oReport AS OBJECT)
    DIM stUrl AS STRING
    DIM arg(0) AS NEW com.sun.star.beans.PropertyValue
    arg(0).name = "FilterName"
    arg(0).value = "writer_pdf_Export"
    stUrl = "file:///...."
    oReport.storeToURL(stUrl, arg())
End Sub
```

Bei der URL muss natürlich eine komplette URL-Adresse angegeben werden. Noch sinnvoller ist es, diese Adresse z.B. gekoppelt mit einem unverwechselbaren Merkmal des gedruckten Dokumentes zu versehen, wie z.B. der Rechnungsnummer. Sonst könnte es passieren, dass eine Sicherungsdatei beim nächsten Druck einfach überschrieben wird.

Druck von Berichten aus einem externen Formular heraus

Schwierig wird es, wenn mit externen Formularen gearbeitet wird. Die Berichte liegen dann in der *.odb-Datei und sind auch über den Datenquellenbrowser erst einmal nicht verfügbar.

```
SUB Berichtsstart(oEvent AS OBJECT)
    DIM oFeld AS OBJECT
    DIM oForm AS OBJECT
    DIM oDocument AS OBJECT
    DIM oDocView AS OBJECT
    DIM Arg()
    oFeld = oEvent.Source.Model
    oForm = oFeld.Parent
    sURL = oForm.DataSourceName
    oDocument = StarDesktop.loadComponentFromURL(sURL, "_blank", 0, Arg() )
    oDocView = oDocument.CurrentController.Frame.ContainerWindow
    oDocView.Visible = False
    oDocument.getCurrentController().connect
    Wait(100)
    oDocument.ReportDocuments.getByName("Bericht").open
    oDocument.close(True)
END SUB
```

Der Bericht wird von einem Button des externen Formulars gestartet. Über den Button wird das Formular ermittelt, in dem der Pfad zur *.odb-Datei verzeichnet ist: **`oForm.DataSourceName`**. Anschließend wird mit **`loadComponentFromUrl`** die *.odb-Datei geöffnet. Die Datei soll nur im Hintergrund liegen. Deshalb wird gleich auf die Ansicht zugegriffen und die Oberfläche der *.odb-Datei auf **`Visible = False`** gestellt. Dies sollte auch direkt beim Aufruf über die Argumentenliste **`Arg()`** funktionieren, brachte aber bei Tests nicht den gewünschten Erfolg.

Wird jetzt direkt der Bericht des geöffneten Dokumentes aufgerufen, so ist die Datenbankverbindung noch nicht verfügbar. Der Bericht erscheint nur mit einem grauen Hintergrund und LibreOffice verzeichnet einen Absturz. Schon eine kleine Wartezeit von 100 Millisekunden (**`Wait(100)`**) löst dieses Problem. Hier müssen praktische Tests zeigen, wie kurz diese Zeit eingestellt werden kann. Anschließend wird der Bericht gestartet. Da es sich bei dem ausgeführten Bericht um eine separate Textdatei handelt, kann die geöffnete *.odb-Datei anschließend geschlossen werden. Mit **`oDocument.close(True)`** wird der Befehl an die *.odb-Datei weiter gegeben. Die Datei wird allerdings erst dann geschlossen, wenn sie nicht mehr aktiv z.B. Daten an die Berichtsdatei weiter geben muss.

Mit einem entsprechenden Zugriff können auch die Formulare innerhalb der *.odb-Datei gestartet werden. Hier sollte dann aber das Schließen des Dokumentes unterbleiben.

Deutlich schneller als mit dem Report-Builder und trotzdem gut gestaltet geht der Druck aber über Makros mit Hilfe von Serienbrieffunktionen oder Textfeldern.

Serienbriefdruck aus Base heraus

Manchmal reicht einfach ein Bericht nicht aus, um sauber Briefe an die Adressaten zu erstellen. Schon die Textfelder in einem Bericht sind hier in der Nutzung doch sehr eingeschränkt. Hierzu wird dann ein Serienbrief im Writer erstellt. Es muss aber nicht sein, dass erst der Writer geöffnet wird, dort dann über den Serienbriefdruck alle Auswahlmöglichkeiten und Eingaben gemacht werden und schließlich der Druck erfolgt. Dies alles geht auch per Makro direkt aus Base heraus.

```
SUB Serienbriefdruck
    DIM oMailMerge AS OBJECT
    DIM aProps()
    oMailMerge = createunoservice("com.sun.star.text.MailMerge")
```

Als Name der Datenquelle wird der Name angegeben, unter dem die Datenbank in LO angemeldet ist. Dieser Name muss nicht identisch mit dem Dateinamen sein. Der Anmeldename in diesem Beispiel lautet "Adressen"

```
    oMailMerge.DataSourceName = "Adressen"
```

Die Pfadbeschreibung mit der Serienbriefdatei erfolgt in der Art der jeweiligen Betriebssystemumgebung, hier ab dem Wurzelpfad eines Linux-Systems.

```
oMailMerge.DocumentURL = ConvertToUrl("home/user/Dokumente/Serienbrief.odt")
```

Der Typ des Kommandos wird festgelegt. '0' steht für eine Tabelle, '1' für eine Abfrage und '2' für ein direktes SQL-Kommando.

```
oMailMerge.CommandType = 1
```

Hier wurde eine Abfrage gewählt, die den Namen "Serienbriefabfrage" trägt.

```
oMailMerge.Command = "Serienbriefabfrage"
```

Über den Filter wird festgelegt, für welche Datensätze aus der Serienbriefabfrage ein Druck erfolgen soll. Dieser Filter könnte z.B. über ein Formularfeld aus Base heraus an das Makro weitergegeben werden. Mit dem Primärschlüssel eines Datensatzes könnte so der Ausdruck eines einzelnen Dokumentes erfolgen.

In diesem Beispiel wird aus der "Serienbriefabfrage" das Feld "Geschlecht" aufgesucht und dort nach Datensätzen gesucht, die in diesem Feld mit einem 'm' versehen sind.

```
oMailMerge.Filter = """Geschlecht""='m'"
```

Es gibt die Ausgabetypen Drucker (1), Datei (2) und Mail (3). Hier wurde zu Testzwecken die Ausgabe in eine Datei gewählt. Diese Datei wird in dem angegebenen Pfad abgespeichert. Für jeden Serienbriefdatensatz wird ein Druck erzeugt. Damit dieser Druck unterscheidbar ist, wird das Feld Nachname in den Dateinamen aufgenommen.

```
    oMailMerge.OutputType = 2
    oMailMerge.OutputUrl = ConvertToUrl("home/user/Dokumente")
    oMailMerge.FileNameFromColumn = True
    oMailMerge.Filenameprefix = "Nachname"
    oMailMerge.execute(aProps())
END SUB
```

Wird allein der Filter über ein Formular bestückt, so kann auf diese Art und Weise, also ohne die Öffnung des Writer-Dokumentes, ein Serienbriefdruck erfolgen.

Drucken über Textfelder

Über **Einfügen → Feldbefehl → Funktionen → Platzhalter** wird im Writer eine Vorlage für das zukünftig zu druckende Dokument erstellt. Die Platzhalter sollten dabei sinnvollerweise mit dem Namen versehen werden, den die Felder auch in der Datenbank bzw. der Tabelle/Abfrage für das Formular haben, aus dem heraus das Makro aufgerufen wird.

Für einfache Zwecke wird «Text» als Typ für den Platzhalter gewählt.

In dem Makro wird der Pfad zur Vorlage hinterlegt. Es wird ein neues Dokument «Unbenannt1.odt» erstellt. Vom Makro werden die Platzhalter über die Abfrage des Inhaltes des aktuellen Datensatzes des Formulars befüllt. Das offene Dokument kann nun noch nach Belieben verändert werden.

In der Beispieldatenbank «Beispiel_Datenbank_Serienbrief_direkt.odb» wird gezeigt, wie mit Hilfe von Textfeldern und einem Zugriff auf eine in der Vorlage bereits vorgesehenen Tabelle eine komplette Rechnung erstellt werden kann. Im Gegensatz zum Report-Builder sind bei dieser Form der Rechnungserstellung die entsprechenden Felder für den Tabelleninhalt nicht in der Höhe begrenzt. Deshalb wird immer aller Text angezeigt.

Hier Teile des Codes, der im Wesentlichen diesem Beitrag von DPunch zu verdanken ist: *http://de.openoffice.info/viewtopic.php?f=8&t=45868#p194799*

```
SUB Textfelder_Fuellen
    oForm = thisComponent.Drawpage.Forms.MainForm
    IF oForm.RowCount = 0 THEN
        msgbox "Kein Datensatz zum Drucken vorhanden"
        EXIT SUB
    END IF
```

Das Hauptformular wird angesteuert. Hier könnte auch die Lage des auslösenden Buttons das Formular selbst ermitteln. Anschließend wird geklärt, ob in dem Formular überhaupt Daten liegen, die einen Druck ermöglichen.

```
oColumns = oForm.Columns
oDB = ThisComponent.Parent
```

Der Zugriff auf die URL ist nicht vom Formular aus direkt möglich. Es muss auf den darüber liegenden Frame der Datenbank Bezug genommen werden.

```
stDir = Left(oDB.Location,Len(oDB.Location)-Len(oDB.Title))
```

Der Titel der Datenbank wird von der URL abgetrennt.

```
stDir = stDir & "Beispiel_Textfelder.ott"
```

Die Vorlage wird aufgesucht und geöffnet

```
DIM args(0) AS NEW com.sun.star.beans.PropertyValue
args(0).Name = "AsTemplate"
args(0).Value = True
oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
```

Die Textfelder werden eingelesen

```
oTextfields = oNewDoc.Textfields.createEnumeration
DO WHILE oTextfields.hasMoreElements
   oTextfield = oTextfields.nextElement
   IF oTextfield.supportsService("com.sun.star.text.TextField.JumpEdit") THEN
      stColumnname = oTextfield.PlaceHolder
```

Placeholder ist die Benennung für das Textfeld

```
         IF oColumns.hasByName(stColumnname) THEN
```

Wenn der Name des Textfeldes gleich dem Spaltennamen der Daten ist, die dem Formular zugrunde liegen, wird der Inhalt aus der Datenbank auf das Feld in dem Textdokument übertragen.

```
            inIndex = oForm.findColumn(stColumnname)
            oTextfield.Anchor.String = oForm.getString(inIndex)
         END IF
      END IF
   LOOP
END SUB
```

Aufruf von Anwendungen zum Öffnen von Dateien

Mit dieser Prozedur kann durch einen Mausklick in ein Textfeld ein Programm aufgerufen werden, das im eigenen Betriebssystem mit der Dateinamensendung verbunden ist. Damit werden auch Links ins Internet oder sogar das Öffnen des Mailprogramms mit einer bestimmten Mailadresse möglich, die gerade in der Datenbank gespeichert wurde.

Siehe zu diesem Abschnitt auch die **Beispieldatenbank** «Beispiel_Mailstart_Dateiaufruf.odb»[14]

```
SUB Link_oeffnen
   DIM oDoc AS OBJECT
   DIM oDrawpage AS OBJECT
   DIM oForm AS OBJECT
   DIM oFeld AS OBJECT
   DIM oShell AS OBJECT
   DIM stFeld AS STRING
   oDoc = thisComponent
   oDrawpage = oDoc.Drawpage
   oForm = oDrawpage.Forms.getByName("Formular")
   oFeld = oForm.getByName("Adresse")
```

14 In der Datenbank «Beispiel_Formular_Eingabekontrolle.odb» ist hierzu eine Prozedur enthalten, die alle Anwendungen öffnet, die irgendwie als Dateiendung mit dem System verbunden sind: Webseiten, Email-Programme, Bilddateien, Textdateien ...

Aus dem benannten Feld wird der Inhalt ausgelesen. Dies kann eine Webadresse, beginnend mit **'http://'**, eine Mailadresse mit einem **'@'** oder ein einfaches Dokument sein, das durch eine entsprechende Pfadangabe aufgesucht werden soll (z.B. externe Bilder, *.pdf-Dateien, Tondokumente …).

```
    stFeld = oFeld.Text
    IF stFeld = "" THEN
        EXIT SUB
    END IF
```

Ist das Feld leer, so soll das Makro sofort enden. Bei der Eingabe passiert es ja sehr oft, dass Felder mit der Maus aufgesucht werden. Ein Mausklick in das Feld, um dort zum ersten Mal schreiben zu können, soll aber noch nicht das Makro ausführen.

Jetzt wird gesucht, ob in dem Feld ein **'@'** enthalten ist. Dies deutet auf eine E-Mail-Adresse hin. Es soll also das Mailprogramm gestartet werden und eine Mail an diese Mailadresse gesandt werden.

```
    IF InStr(stFeld,"@") THEN
        stFeld = "mailto:"+stFeld
```

Ist kein **'@'** vorhanden, so wird der Begriff so konvertiert, dass die Datei im Dateisystem gefunden werden kann. Steht ein **'http://'** am Anfang, so wird bei dieser Funktion nicht im Dateisystem, sondern über den Webbrowser direkt im Internet nachgesehen. Ansonsten beginnt der erstellte Pfad anschließend mit dem Begriff **'file:///'**

```
    ELSE
        stFeld = convertToUrl(stFeld)
    END IF
```

Bei der Verwendung von Sonderzeichen in URLs kann es sinnvoll sein, die Konvertierung für den Pfad zu unterlassen. Der Shell-Befehl funktioniert auch mit der systeminternen Schreibweise. Hier müsste dann allerdings separat für die Endungen **'http://'** und **'https://'** eine Konvertierung vorgenommen werden. Jetzt wird das Programm aufgesucht, das in dem eigenen Betriebssystem mit der entsprechenden Dateiendung verbunden ist. Bei dem Stichwort **'mailto:'** ist dies das Mailprogramm, bei **'http://'** der Webbrowser und bei allen anderen ist die Entscheidung des Systems mit den Endungen der Datei verbunden.

```
    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
    oShell.execute(stFeld,,0)
END SUB
```

Aufruf eines Mailprogramms mit Inhaltsvorgaben

Eine Erweiterung des vorhergehenden Beispiels zum Programmaufruf stellt dieser Aufruf eines Mailprogramms mit Vorgaben in der Betreffzeile und inhaltlichen Vorgaben dar.

Siehe auch zu diesem Abschnitt die **Beispieldatenbank** «Mailstart_Dateiaufruf.odb»

Der Mailaufruf erfolgt mit **'mailto:Empfänger?subject= &body= &cc= &bcc= '**. Die letzten beiden Eingaben sind im Formular allerdings nicht aufgeführt. Anhänge kommen in der Definition von **'mailto'** nicht vor. Manchmal funktioniert allerdings **'attachment='**.

```
SUB Mail_Aufruf
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM oFeld2 AS OBJECT
    DIM oFeld3 AS OBJECT
    DIM oFeld4 AS OBJECT
    DIM oShell AS OBJECT
    DIM stFeld1 AS STRING
    DIM stFeld2 AS STRING
```

```
    DIM stFeld3 AS STRING
    DIM stFeld4 AS STRING
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("Formular")
    oFeld1 = oForm.getByName("E_Mail_Adresse")
    oFeld2 = oForm.getByName("E_Mail_Betreff")
    oFeld3 = oForm.getByName("E_Mail_Inhalt")
    stFeld1 = oFeld1.Text
    IF stFeld1 = "" THEN
        msgbox "Keine Mailadresse vorhanden." & CHR(13) &
            "Das Mailprogramm wird nicht aufgerufen" , 48, "Mail senden"
        EXIT SUB
    END IF
```

Die Konvertierung zu URL ist notwendig, damit Sonderzeichen und Zeilenumbrüche den Aufruf nicht stören. Dabei wird allerdings auch der Begriff **`'file:///'`** vorangestellt. Diese 8 Zeichen zu Beginn werden nicht mit übernommen.

```
    stFeld2 = Mid(ConvertToUrl(oFeld2.Text),9)
    stFeld3 = Mid(ConvertToUrl(oFeld3.Text),9)
```

Im Gegensatz zum einfachen Programmaufruf werden hier Details des Mailaufrufes über den Aufruf des Mailprogramms mitgegeben.

```
    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
    oShell.execute("mailto:" + stFeld1 + "?subject=" + stFeld2 + "&body=" + stFeld3,,0)
END SUB
```

Hinweis

Das Versenden von Mails mit Hilfe des Mailprogramms kann auch mit folgendem Code erfolgen. Allerdings kann der eigentliche Inhalt der Mail hier nicht mit eingefügt werden.

```
  DIM attachs(0)
  IF GetGuiType() = 1 THEN
     oMailer = createUnoService("com.sun.star.system.SimpleSystemMail")
     ' Sonst Linux/Mac
     ELSE
     oMailer = createUnoService("com.sun.star.system.SimpleCommandMail")
  END IF
  oMailProgramm = oMailer.querySimpleMailClient()
  oNeueNachricht = oMailProgramm.createSimpleMailMessage()
  oNeueNachricht.setRecipient(stFeld1)
  oNeueNachricht.setSubject(stFeld2)
  attachs(0) = "file:///..."
  oNeueNachricht.setAttachement(attachs())
  oMailprogramm.sendSimpleMailMessage(oNeuenachricht, 0 )
```

Zu den möglichen Parametern siehe: *http://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1system_1_1XSimpleMailMessage.html*

Aufruf einer Kartenansicht zu einer Adresse

Eine Datenbank enthält lauter Adressen. Jetzt soll zu einer Adresse aufgezeigt werden, in welcher Umgebung denn das Haus liegt. Die folgende Prozedur «MapPosition» wird mit einem Button gestartet, der in dem gleichen Formular liegt, in dem die Angaben zur Adresse verzeichnet sind.[15]

```
SUB MapPosition(oEvent AS OBJECT)
    DIM oForm AS OBJECT, oShell AS OBJECT
    DIM i AS INTEGER
    DIM stLink AS STRING, stTag AS STRING
    DIM arFields()
    stTag = oEvent.Source.Model.Tag
    oForm = oEvent.Source.Model.Parent
```

15 Siehe Beispiel_Formular_Eingabekontrolle.odb

```
    arFields = Split(stTag,",")
```

In den Zusatzinformationen des Buttons sind, durch Kommas getrennt, die Namen der Felder aufgeführt, die zusammen die Adresse ergeben. Dies sind in der Beispieldatenbank **comPLZOrt,txtStraße**. Das erste Feld ist ein Kombinationsfeld, das die Postleitzahl und den Ort enthält, das zweite Feld enthält die Straße und die Hausnummer. Die beiden Feldbezeichnungen werden voneinander getrennt und in ein Array geschrieben.

```
    FOR i = LBound(arFields) TO UBound(arFields)
        IF stLink = "" THEN
            stLink = oForm.getByName(arFields(i)).CurrentValue
        ELSE
        stLink = stLink & "+" & oForm.getByName(arFields(i)).CurrentValue
        END IF
    NEXT i
```

Die Inhalte der beiden Felder werden ausgelesen und mit einem **+** verbunden in der Variablen **stLink** gespeichert. Dieser Suchstring wird jetzt in den Link für nominatim.openstreetmap.org eingefügt. Beim Einfügen wird darauf geachtet, dass auch die Leerzeilen in dem String mit **+** ausgefüllt werden. Dies geschieht, indem der String einfach einmal an den Leerzeichen durch **Split** aufgetrennt wird und dann wieder über **Join** mit einem **+** die Teile verbunden werden.

```
    IF stLink <> "" THEN
        stLink = "https://nominatim.openstreetmap.org/search.php?q=" &
            Join(Split(stLink),"+") & "&polygon_geojson=1&viewbox="
        oShell = createUnoService("com.sun.star.system.SystemShellExecute")
        oShell.execute(stLink,,0)
    END IF
END SUB
```

Die weiteren Elemente des Links sind lediglich aus dem Link entstanden, den die Website bei direkter Nutzung der Suchfunktion angibt. Wie in den vorhergehenden Beispielen wird dieser Link über die **SystemShell** gestartet. Dort wird dann der Browser aufgerufen, der bei einer in der Karte verzeichneten Adresse die auch direkt findet.[16]

Mauszeiger beim Überfahren eines Links ändern

Im Internet üblich, bei Base nachgebaut: Der Mauszeiger fährt über ein Textfeld und verändert seine Form zu einer zeigenden Hand. Der enthaltene Text kann jetzt noch in den Eigenschaften des Feldes zu der Farbe Blau und unterstrichen geändert werden – schon ist der Eindruck eines Links aus dem Internet perfekt. Jeder Nutzer erwartet nach einem Klick, dass sich ein externes Programm öffnet.

Siehe auch zu diesem Abschnitt die **Beispieldatenbank** «Mailstart_Dateiaufruf.odb»

Diese kurze Prozedur sollte mit dem Ereignis **'Maus innerhalb'** des Textfeldes verbunden werden.

```
SUB Mauszeiger(Event AS OBJECT)
    REM Siehe auch Standardbibliotheken: Tools → ModuleControls → SwitchMousePointer
    DIM oPointer AS OBJECT
    oPointer = createUnoService("com.sun.star.awt.Pointer")
    oPointer.setType(27)'Typen in com.sun.star.awt.SystemPointer
    Event.Source.Peer.SetPointer(oPointer)
END SUB
```

Formulare ohne Symbolleisten präsentieren

Neunutzer von Base sind häufig irritiert, dass z.B. eine Menüleiste existiert, diese aber im Formular so gar nicht verfügbar ist. Diese Menüleisten können auf verschiedene Arten ausgeblendet wer-

16 Bei Nutzung dieser Möglichkeit der Kartendarstellung sollten die Bedingungen der Website beachtet werden: *https://operations.osmfoundation.org/policies/nominatim/* .

den. Am erfolgreichsten unter allen LO-Versionen sind die beiden im Folgenden vorgestellten Vorgehensweisen.

Fenstergrößen und Symbolleisten werden in der Regel über ein Makro beeinflusst, das in einem Formulardokument unter **Extras → Anpassen → Ereignisse → Dokument öffnen** gestartet wird. Gemeint ist hier das Dokument, nicht ein einzelnes Haupt- oder Unterformular.

Formulare ohne Symbolleisten in einem Fenster

Ein Fenster lässt sich in der Größe variieren. Über den entsprechenden Button lässt es sich auch schließen. Diese Aufgaben übernimmt der Window-Manager des jeweiligen Betriebssystems. Lage und Größe des Fensters auf dem Bildschirm kann beim Start über ein Makro mitgegeben werden.

```
SUB Symbolleisten_Ausblenden
    DIM oFrame AS OBJECT
    DIM oWin AS OBJECT
    DIM oLayoutMng AS OBJECT
    DIM aElemente()
    oFrame = StarDesktop.getCurrentFrame()
```

Der Titel für das Formular wird in der Titelleiste des Fensters angezeigt.

```
    oFrame.setTitle "Mein Formular"
    oWin = oFrame.getContainerWindow()
```

Das Fenster wird auf die maximale Größe eingestellt. Dies entspricht nicht dem Vollbildmodus, da z.B. eine Kontrollleiste noch sichtbar ist und das Fenster eine Titelleiste hat, über die die Größe des Fensters geändert und das Fenster geschlossen werden kann.

```
    oWin.IsMaximized = true
```

Es besteht auch die Möglichkeit, das Fenster in einer ganz bestimmten Größe und mit einer festen Position darzustellen. Dies würde mit **`'oWin.setPosSize(0,0,600,400,15)'`** geschehen. Hier wird das Fenster an der linken oberen Ecke des Bildschirms mit einer Breite von 600 Punkten und einer Höhe von 400 Punkten dargestellt. Die letzte Ziffer weist darauf hin, dass alle Punkte angegeben wurden. Sie wird als **`'Flag'`** bezeichnet. Das **`'Flag'`** wird aus den folgenden Werten über eine Summierung berechnet: x=1, y=2, Breite=4, Höhe=8. Da x, y, Breite und Höhe angegeben sind, hat das **`'Flag'`** die Größe 1 + 2 + 4 + 8 = 15.

```
    oLayoutMng = oFrame.LayoutManager
    aElemente = oLayoutMng.getElements()
    FOR i = LBound(aElemente) TO UBound(aElemente)
        IF aElemente(i).ResourceURL =
            "private:resource/toolbar/formsnavigationbar" THEN
        ELSE
            oLayoutMng.hideElement(aElemente(i).ResourceURL)
        END IF
    NEXT
    ThisComponent.CurrentController.Sidebar.Visible = False
END SUB
```

Wenn es sich um die Navigationsleiste handelt, soll nichts geschehen. Das Formular soll schließlich bedienbar bleiben, wenn nicht das Kontrollfeld für die Navigationsleiste eingebaut und die Navigationsleiste sowieso ausgeblendet wurde. Nur wenn es sich nicht um die **Navigationsleiste** handelt, soll die entsprechende Leiste verborgen werden. Deswegen erfolgt zu dieser Bedingung **keine Aktion**.

Werden die Symbolleisten nicht wieder direkt beim Beenden des Formulars eingeblendet, so bleiben sie weiterhin verborgen. Sie können natürlich über **Ansicht → Symbolleisten** wieder aufgerufen werden. Etwas irritierend ist es jedoch, wenn gerade die Standardleiste (**Ansicht → Symbolleisten → Standardleiste**) oder die Statusleiste (**Ansicht → Statusleiste**) fehlt.

Mit dieser Prozedur werden die Symbolleisten aus dem Versteck (**`'hideElement'`**) wieder hervorgeholt (**`'showElement'`**). Der Kommentar enthält die Leisten, die oben als sonst fehlende Leisten am ehesten auffallen.

```
SUB Symbolleisten_Einblenden
    DIM oFrame AS OBJECT
    DIM oLayoutMng AS OBJECT
    DIM aElemente()
    oFrame = StarDesktop.getCurrentFrame()
    oLayoutMng = oFrame.LayoutManager
    aElemente = oLayoutMng.getElements()
    FOR i = LBound(aElemente) TO UBound(aElemente)
       oLayoutMng.showElement(aElemente(i).ResourceURL)
    NEXT
    ' eventuell fehlende wichtige Elemente:
    '   "private:resource/toolbar/standardbar"
    '   "private:resource/statusbar/statusbar"
    ThisComponent.CurrentController.Sidebar.Visible = True
END SUB
```

Die Makros werden an die Eigenschaften des Formularfensters gebunden: **Extras → Anpassen → Ereignisse → Dokument öffnen → Symbolleisten_Ausblenden** bzw. **… Dokument geschlossen → Symbolleisten_Einblenden**

Leider tauchen häufig Symbolleisten trotzdem nicht wieder auf. In hartnäckigen Fällen kann es daher helfen, nicht die Elemente auszulesen, die der Layoutmanager bereits kennt, sondern definitiv bestimmte Symbolleisten erst zu erstellen und danach schließlich zu zeigen:

```
Sub Symbolleisten_Einblenden
    DIM oFrame AS OBJECT
    DIM oLayoutMng AS OBJECT
    DIM i AS INTEGER
    DIM aElemente(5) AS STRING
    oFrame = StarDesktop.getCurrentFrame()
    oLayoutMng = oFrame.LayoutManager
    aElemente(0) = "private:resource/menubar/menubar"
    aElemente(1) = "private:resource/statusbar/statusbar"
    aElemente(2) = "private:resource/toolbar/formsnavigationbar"
    aElemente(3) = "private:resource/toolbar/standardbar"
    aElemente(4) = "private:resource/toolbar/formdesign"
    aElemente(5) = "private:resource/toolbar/formcontrols"
    FOR EACH i IN aElemente()
       IF NOT(oLayoutMng.requestElement(i)) THEN
          oLayoutMng.createElement(i)
       END IF
    oLayoutMng.showElement(i)
    NEXT i
    ThisComponent.CurrentController.Sidebar.Visible = True
END SUB
```

Die darzustellenden Symbolleisten werden explizit benannt. Ist eine der entsprechenden Symbolleisten nicht für den Layoutmanger vorhanden, so wird sie zuerst über **`createElement`** erstellt und danach über **`showElement`** gezeigt.

Formulare im Vollbildmodus

Beim Vollbildmodus wird der gesamte Bildschirm vom Formular bedeckt. Hier steht keine Kontrollleiste o.ä. mehr zur Verfügung, die gegebenenfalls anzeigt, ob noch irgendwelche anderen Programme laufen.

```
FUNCTION Fullscreen(boSwitch AS BOOLEAN)
    DIM oDispatcher AS OBJECT
    DIM Props(0) AS NEW com.sun.star.beans.PropertyValue
    oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
    Props(0).Name = "FullScreen"
    Props(0).Value = boSwitch
    oDispatcher.executeDispatch(ThisComponent.CurrentController.Frame,
       ".uno:FullScreen", "", 0, Props())
END FUNCTION
```

Diese Funktion wird durch die folgenden Prozeduren eingeschaltet. In den Prozeduren läuft gleichzeitig die vorhergehende Prozedur zum Ausblenden der Symbolleisten ab – sonst erscheint die Symbolleiste, mit der der Vollbildmodus wieder ausgeschaltet werden kann. Auch dies ist eine Symbolleiste, wenn auch nur mit einem Symbol.

```
SUB Vollbild_ein
    Fullscreen(true)
    Symbolleisten_Ausblenden
END SUB
```

Aus dem Vollbild-Modus geht es wieder heraus über die **'ESC'**-Taste. Wenn stattdessen ein Button mit einem entsprechenden Befehl belegt werden soll, so reichen auch die folgenden Zeilen:

```
SUB Vollbild_aus
    Fullscreen(false)
    Symbolleisten_Einblenden
END SUB
```

Formular direkt beim Öffnen der Datenbankdatei starten

Wenn jetzt schon die Symbolleisten weg sind oder gar das Formular im Vollbildmodus erscheint, dann müsste nur noch die Datenbankdatei beim Öffnen direkt in dieses Formular hinein starten. Der einfache Befehl zum Öffnen von Formularen reicht dabei leider nicht aus, da die Datenbankverbindung beim Öffnen des Base-Dokumentes noch nicht besteht.

Das folgende Makro wird über **Extras → Anpassen → Ereignisse → Dokument öffnen** gestartet. Dabei ist **Speichern in → Datenbankdatei.odb** zu wählen.

```
SUB Formular_Direktstart
    DIM oDatenquelle AS OBJECT
    oDatenquelle = ThisDatabaseDocument.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
       oDatenquelle.connect()
    END IF
    ThisDatabaseDocument.FormDocuments.getByName("Formularname").open
    REM alternativ geht auch:
    'oDatenquelle.loadComponent(com.sun.star.sdb.application.DatabaseObject.FORM,
    "Formularname",FALSE)
END SUB
```

Zuerst muss der Kontakt mit der Datenquelle hergestellt werden. Der Controller hängt ebenso mit **ThisDatabaseDocument** zusammen wie das Formular. Anschließend kann das Formular gestartet werden und liest auch die Datenbankinhalte aus.

MySQL-Datenbank mit Makros ansprechen

Sämtliche bisher vorgestellten Makros wurden mit der internen HSQLDB verbunden. Bei der Arbeit mit externen Datenbanken sind ein paar Änderungen und Erweiterungen notwendig.

MySQL-Code in Makros

Wird die interne Datenbank angesprochen, so werden die Tabellen und Felder mit doppelten Anführungszeichen gegenüber dem SQL-Code abgesetzt:

SELECT "Feld" FROM "Tabelle"

Da in Makros der SQL-Befehl Text darstellt, müssen die doppelten Anführungszeichen zusätzlich maskiert werden:

```
stSQL = "SELECT ""Feld"" FROM ""Tabelle"""
```

MySQL-Abfragen hingegen werden anders maskiert:

SELECT `Feld` FROM `Datenbank`.`Tabelle`

Durch diese andere Form der Maskierung wird daraus im Makro-Code:

```
stSql = "SELECT `Feld` FROM `Datenbank`.`Tabelle`"
```

Temporäre Tabelle als individueller Zwischenspeicher

In den vorhergehenden Kapiteln wurde häufiger eine einzeilige Tabelle zum Suchen oder Filtern von Tabellen genutzt. In einem Mehrbenutzersystem kann darauf nicht zurückgegriffen werden, da sonst andere Nutzer von dem Filterwert eines anderen Nutzers abhängig würden. Temporäre Tabellen sind in MySQL nur für den Nutzer der gerade aktiven Verbindung zugänglich, so dass für die Such- und Filterfunktionen auf diese Tabellenform zugegriffen werden kann.

Diese Tabellen können natürlich nicht vorher erstellt worden sein. Sie müssen beim Öffnen der Base-Datei erstellt werden. Deshalb ist das folgende Makro mit dem Öffnen der *.odb-Datei zu verbinden:

```
SUB CreateTempTable
    oDatenquelle = thisDatabaseDocument.CurrentController
    IF NOT (oDatenquelle.isConnected()) THEN oDatenquelle.connect()
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "CREATE TEMPORARY TABLE IF NOT EXISTS `Suchtmp` (`ID` INT PRIMARY KEY,
        `Name` VARCHAR(50))"
    oSQL_Anweisung.executeUpdate(stSql)
END SUB
```

Zum Start der *.odb-Datei besteht noch keine Verbindung zur externen MySQL-Datenbank. Die Verbindung muss erst einmal hergestellt werden. Dann wird eine temporäre Tabelle mit entsprechend notwendigen Feldern erstellt.

Leider zeigt Base die temporären Tabellen nicht im Tabellencontainer an. Es kann über Abfragen auf diese Tabellen zugegriffen werden. Der Zugriff ist allerdings nur lesend möglich, so dass neue Inhalte für diese Tabellen nur über die direkte SQL-Eingabe oder über Makros erfolgen kann. Für einen einfachen Filterzugriff bietet sich deshalb an, statt einer temporären Tabelle eine feste Tabelle zu nutzen, in der die Filterinhalte zusammen mit der Verbindungsnummer (**`CONNECTION_ID`**) gespeichert werden.

Filterung über die Verbindungsnummer

Hier wird die Filtertabelle bereits vorher über die GUI erstellt. Die Tabelle wird beim Öffnen der Datenbankdatei allerdings direkt mit entsprechendem Inhalt versorgt:

```
    stSql = "REPLACE INTO `Filter` (`Connection_ID`,`Name`)
       VALUES(CONNECTION_ID(),NULL)"
```

Die Tabelle ist jetzt auch in Formularen beschreibbar und kann entsprechend einfacher genutzt werden. Für andere Nutzer ist jetzt allerdings sichtbar, nach welchen Begriffen der einzelnen Nutzer gerade sucht. Prinzipiell lässt sich aber der entsprechende auf den einzelnen Nutzer festgelegte Datensatz immer über **`CONNECTION_ID()`** ermitteln.

Wird die Datenbankdatei wieder geschlossen, so kann auch die Filter-Tabelle entsprechend bereinigt werden:

```
SUB DeleteFilter
    oDatasource = thisDatabaseDocument.CurrentController
    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
    oConnection = oDatasource.ActiveConnection()
    oSQL_Command = oConnection.createStatement()
    stSql = "DELETE FROM `Filter` WHERE `Connection_ID` = CONNECTION_ID()"
    oSQL_Command.executeUpdate(stSql)
END SUB
```

Gespeicherte Prozeduren

In MySQL/MariaDB können Prozeduren gespeichert werden. Sollen diese Prozeduren zu bestimmten Zeiten ablaufen, so können sie über **Extras → SQL** mit dem Befehl **`CALL `Prozedurname`();`** aufgerufen werden. Erstellen solche Prozeduren von sich aus eine Ergebnismenge in einer temporären Tabelle, so lässt sich diese temporäre Tabelle als nicht bearbeitbare Informationsquelle nutzen.

Hinweis	Auch hier gilt wie im Kapitel «Datenbank erstellen» der Hinweis, dass die direkte Verbindung mit MySQL/MaraiDB hier einen Bug hat, der den erneuten Aufruf einer Prozedur verhindert und damit die Arbeit mit Prozeduren unmöglich macht. Die JDBC-Verbindung zur MySQL- bzw. MariaDB ist davon nicht betroffen. Vor der Nutzung von gespeicherten Prozeduren sollte also wenigstens unter **Extras → SQL** zweimal hintereinander die Prozedur mit CALL … testweise abgerufen werden.

Automatischer Aufruf einer Prozedur

Die folgende Prozedur **`AlleNamen()`** könnte beim Laden eines Formulars ausgelöst werden. Sie muss ablaufen, bevor das Formular selbst Inhalt laden will. Kann das nicht erfolgen, so muss zusätzlich auf das auslösende Formular über das Ereignis Bezug genommen werden und das Formular nach der Ausführung der Prozedur erneut geladen werden.

```
SUB ProcExecute
    oDatasource = thisDatabaseDocument.CurrentController
    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
    oConnection = oDatasource.ActiveConnection()
    oSQL_Command = oConnection.createStatement()
    oSql_Command.executeUpdate("CALL `AlleNamen`();")
END SUB
```

Die Prozedur ersetzt lediglich den Umweg, das Kommando **`CALL `AlleNamen`();`** über **Extras → SQL** eingeben zu müssen. Die Prozedur wird ohne Rückgabewert genutzt. Der Rückgabewert muss per SQL in der Prozedur selbst definiert sein.

Übertragung der Ausgabe einer Prozedur in eine temporäre Tabelle

Dieses Makro geht davon aus, dass die gespeicherte Prozedur von MySQL/MariaDB einen Rückgabewert hat, der aber leider nicht über eine Abfrage, sondern nur direkt über SQL auf der Konsole direkt ausgegeben wird.

```
SUB ProcContentShow
    oDatasource = thisDatabaseDocument.CurrentController
    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
    oConnection = oDatasource.ActiveConnection()
    oSQL_Command = oConnection.createStatement()
    oResult = oSql_Command.executeQuery("CALL `AlleNamen`();")
    stFields = ""
    FOR i = 1 TO oResult.Columns.Count
        stFields = stFields + "`" + oResult.Columns.ElementNames(i-1) + "` TINYTEXT,"
    NEXT
    stFields = Left(stFields, Len(stFields)-1)
    stProcedure = "("
    WHILE oResult.next
        FOR i = 1 TO oResult.Columns.Count
            stProcedure = stProcedure + "'" + oResult.getString(i) +"',"
        NEXT
        stProcedure =  Left(stProcedure, Len(stProcedure)-1)
        stProcedure = stProcedure + "),("
    WEND
    stProcedure = Left(stProcedure, Len(stProcedure)-2)
    oSQL_Command.executeUpdate("DROP TEMPORARY TABLE IF EXISTS `TempNamen`")
    oSQL_Command.executeUpdate("CREATE TEMPORARY TABLE `TempNamen` ("+stFields+")")
```

```
    oSQL_Command.executeUpdate("INSERT INTO `TempNamen` VALUES "+stProcedure+";" )
END SUB
```

Zuerst wird die Prozedur ausgeführt. Ein eventueller Rückgabewert wird in **`oResult`** gespeichert. Aus diesem Rückgabewert lassen sich die Spaltennamen (**`oResult.Columns.ElementNames()`**) und der Inhalt (**`oResult.getString()`**) auslesen. Die Feldtypen sind leider nicht zu ermitteln, so dass der Inhalt jeder Spalte einfach als Text interpretiert wird. Dieser Text wird als **`TINYTEXT`** mit einer Maximallänge von 255 Zeichen anschließend in einer temporären Tabelle gespeichert. Diese Tabelle kann dann zum Recherchieren genutzt werden.

Dialoge

Statt Formularen können für Base auch Dialoge zur Eingabe von Daten, zum Bearbeiten von Daten oder auch zur Wartung der Datenbank genutzt werden. Dialoge lassen sich auf das jeweilige Anwendungsgebiet direkt zuschneiden, sind aber natürlich nicht so komfortabel vordefiniert wie Formulare. Hier eine kurze Einführung, die mit einem recht komplexen Beispiel zur Datenbankwartung endet.

Dialoge starten und beenden

Zuerst muss für den Dialog[17] ein entsprechender Ordner erstellt werden. Dies geschieht über **Extras → Makros → Dialoge verwalten → Datenbankdatei → Standard → Neu**. Der Dialog erscheint mit einer grauen Fläche und einer Titelleiste sowie einem Schließkreuz. Bereits dieser leere Dialog könnte jetzt aufgerufen und über das Schließkreuz wieder geschlossen werden.

Wird der Dialog angeklickt, so gibt es bei den allgemeinen Eigenschaften die Möglichkeit, die Größe und Position einzustellen. Außerdem kann dort der Inhalt des Titels «Dialoge starten» eingegeben werden.

In der am unteren Fensterrand befindlichen Symbolleiste befinden sich die verschiedensten Formular-Steuerelemente. Aus diesen Steuerelementen sind für den abgebildeten Dialog zwei Schaltflächen ausgesucht worden, von denen aus andere Dialoge gestartet werden sollen. Die Bearbeitung des Inhaltes und der Verknüpfung zu Makros ist gleich den Schaltflächen im Formular.

Die Lage der Deklaration der Variablen für den Dialog ist besonders zu beachten. Der Dialog wird als globale Variable gesetzt, damit auf ihn von unterschiedlichen Prozeduren aus zugegriffen werden kann. In diesem Falle ist der Dialog mit der Variablen oDialog0 versehen, weil es noch weitere Dialoge gibt, die einfach entsprechend durchnummeriert wurden.

```
DIM oDialog0 AS OBJECT
```

Zuerst wird die Bibliothek für den Dialog geladen. Sie liegt in dem Verzeichnis «Standard», sofern bei der Erstellung des Dialogs keine andere Bezeichnung gewählt wurde. Der Dialog selbst ist über den Reiter mit der Bezeichnung «Dialog0» in dieser Bibliothek erreichbar. Mit **`Execute()`** wird der Dialog aufgerufen.

17 Die Beispieldatenbank «Beispiel_Dialoge.odb» zu den folgenden Kapiteln ist den Beispieldatenbanken für dieses Handbuch beigefügt.

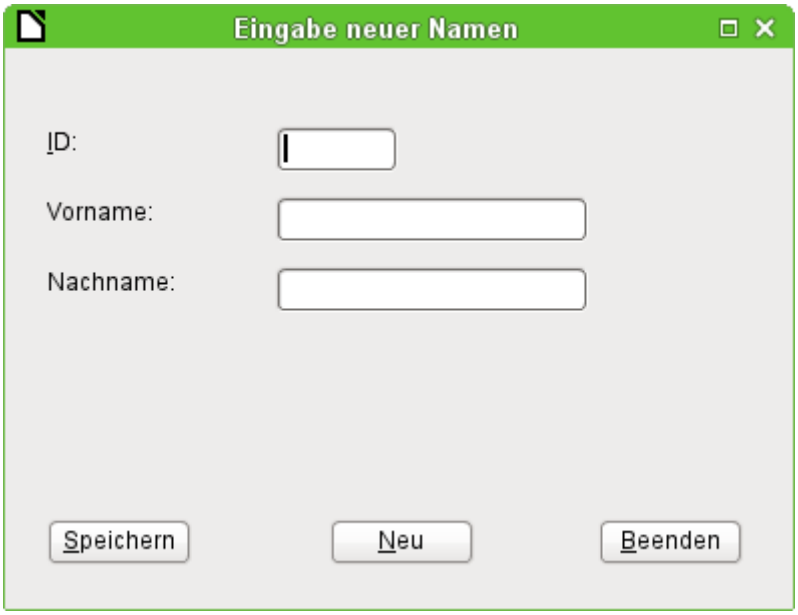
```
SUB Dialog0Start
    DialogLibraries.LoadLibrary("Standard")
    oDialog0 = createUnoDialog(DialogLibraries.Standard.Dialog0)
    oDialog0.Execute()
END SUB
```

Prinzipiell kann ein Dialog durch Betätigung des Schließkreuzes geschlossen werden. Soll dafür aber ein entsprechender Button vorgesehen werden, so reicht hier einfach der Befehl **`EndExecute()`** innerhalb einer Prozedur.

```
SUB Dialog0Ende
    oDialog0.EndExecute()
END SUB
```

Mit diesem Rahmen können beliebig gestaltete Dialoge gestartet und wieder geschlossen werden.

Einfacher Dialog zur Eingabe neuer Datensätze



Dieser Dialog stellt eine Vorstufe für den nächstfolgenden Dialog zur Bearbeitung von Datensätzen dar. Erst einmal sollen nur grundlegende Vorgehensweisen im Umgang mit der Tabelle einer Datenbank geklärt werden. Hier ist dies das Speichern von Datensätzen mit neuem Primärschlüssel bzw. das komplett neue Eingeben von Datensätzen. Wie weit so ein kleiner Dialog ausreichend für eine bestimmte Datenbankaufgabe ist, hängt natürlich von den Bedürfnissen des Nutzer ab.

Mit

```
DIM oDialog1 AS OBJECT
```

wird wieder direkt auf der untersten Ebene des Moduls vor allen Prozeduren die globale Variable für den Dialog erstellt.

Der Dialog wird genauso gestartet und beendet wie der vorhergehende Dialog. Lediglich die Bezeichnung ändert sich von Dialog0 auf Dialog1. Die Prozedur zum Beenden des Dialogs ist mit dem Button Beenden verbunden.

Über den Button Neu werden alle Kontrollfelder des Dialogs durch die Prozedur «DatenfelderLeeren» von vorhergehenden Eingaben befreit:

```
SUB DatenfelderLeeren
    oDialog1.getControl("NumericField1").Text = ""
    oDialog1.getControl("TextField1").Text = ""
    oDialog1.getControl("TextField2").Text = ""
END SUB
```

Jedes Feld, das in einen Dialog eingefügt wird, ist über einen eigenen Namen ansprechbar. Im Gegensatz zu Feldern eines Formulars wird hier durch die Benutzeroberfläche darauf geachtet, dass keine Namen doppelt vergeben werden.

Über **`getControl`** wird zusammen mit dem Namen auf ein Kontrollfeld zugegriffen. Auch ein numerisches Feld hat hier die Eigenschaft **`Text`** zur Verfügung. Nur so lässt sich schließlich ein numerisches Feld leeren. Einen leeren Text gibt es, eine leere Nummer hingegen nicht – stattdessen müsste 0 in das Feld für den Primärschlüssel geschrieben werden.

Der Button Speichern löst schließlich die Prozedur «Daten1Speichern» aus:

```
SUB Daten1Speichern
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM loID AS LONG
    DIM stVorname AS STRING
    DIM stNachname AS STRING
    loID = oDialog1.getControl("NumericField1").Value
    stVorname = oDialog1.getControl("TextField1").Text
    stNachname = oDialog1.getControl("TextField2").Text
    IF loID > 0 AND stNachname <> "" THEN
        oDatenquelle = thisDatabaseDocument.CurrentController
        If NOT (oDatenquelle.isConnected()) THEN
            oDatenquelle.connect()
        END IF
        oVerbindung = oDatenquelle.ActiveConnection()
        oSQL_Anweisung = oVerbindung.createStatement()
        stSql = "SELECT ""ID"" FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        WHILE oAbfrageergebnis.next
            MsgBox ("Der Wert für das Feld 'ID' existiert schon",16,
                "Doppelte Dateneingabe")
            EXIT SUB
        WEND
        stSql = "INSERT INTO ""Name"" (""ID"", ""Vorname"", ""Nachname"")
            VALUES ('"+loID+"','"+stVorname+"','"+stNachname+"')"
        oSQL_Anweisung.executeUpdate(stSql)
        DatenfelderLeeren
    END IF
END SUB
```

Wie in der Prozedur «DatenfelderLeeren» wird auf die Eingabefelder zugegriffen. Dieses Mal erfolgt der Zugriff allerdings lesend, nicht schreibend. Nur wenn das Feld «ID» eine Eingabe größer als 0 enthält und in dem Feld für den Nachnamen auch Text steht, soll der Datensatz weitergegeben werden. Die Null muss alleine schon deshalb ausgeschlossen werden, weil eine Zahlenvariable für Zahlen ohne Nachkommastellen grundsätzlich mit dem Wert 0 initialisiert wird. Auch bei einem leeren Feld würde also schließlich 0 zur Speicherung weitergegeben.

Sind die beiden Felder entsprechend mit Inhalt versehen, so wird eine Verbindung zur Datenbank aufgenommen. Da sich die Kontrollfelder nicht in einem Formular befinden, muss die Datenbankverbindung über **`thisDatabaseDocument.CurrentController`** hergestellt werden.

Zuerst wird eine Abfrage an die Datenbank geschickt, ob vielleicht ein Datensatz mit dem vorgegebenen Primärschlüssel schon existiert. Hat die Abfrage Erfolg, so wird eine Meldung über eine Messagebox ausgegeben, die mit einem Stopp-Symbol versehen ist (Code: **16**) und die Überschrift «Doppelte Dateneingabe» trägt. Danach wird durch **`Exit SUB`** die Prozedur beendet.

Hat die Abfrage keinen Datensatz gefunden, der den gleichen Primärschlüssel hat, so wird der neue Datensatz über den Insert-Befehl in die Datenbank eingefügt. Anschließend wird über die Prozedur «DatenfelderLeeren» wieder ein leeres Formular präsentiert.

Dialog zum Bearbeiten von Daten in einer Tabelle

Dieser Dialog stellt schon deutlich mehr Möglichkeiten zur Verfügung als der vorhergehende Dialog. Hier lassen sich alle Datensätze anzeigen, durch Datensätze navigieren, Datensätze neu erstellen, ändern oder auch löschen. Natürlich ist der Code entsprechend umfangreicher.

Die Button Beenden ist mit der entsprechend auf den Dialog2 abgewandelten Prozedur des vorhergehenden Dialogs zur Eingabe neuer Datensätze verbunden. Hier werden nur die weiteren Buttons mit ihren entsprechenden Funktionen beschrieben.

Die Dateneingabe ist im Dialog so beschränkt, dass im Feld «ID» der Mindestwert auf '1' eingestellt wurde. Diese Einschränkung hat mit dem Umgang mit Variablen in Basic zu tun: Zahlenvariablen werden bei der Definition bereits mit '0' als Grundwert vorbelegt. Werden Zahlenwerte von leeren Feldern und Feldern mit '0' ausgelesen, so ist für Basic der anschließende Inhalt der Variablen gleich. Es müsste bei der Nutzung von '0' im Feld «ID» also zur Unterscheidung erst Text ausgelesen und vielleicht später in eine Zahl umgewandelt werden.

Der Dialog wird unter den gleichen Voraussetzungen geladen wie vorher auch. Hier wird allerdings die Ladeprozedur davon abhängig gemacht, ob die Variable, die der Prozedur «DatenLaden» mitgegeben wird, 0 ist.

```
SUB DatenLaden(loID AS LONG)
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM stVorname AS STRING
    DIM stNachname AS STRING
    DIM loRow AS LONG
    DIM loRowMax AS LONG
    DIM inStart AS INTEGER
    oDatenquelle = thisDatabaseDocument.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    IF loID < 1 THEN
        stSql = "SELECT MIN(""ID"") FROM ""Name"""
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        WHILE oAbfrageergebnis.next
            loID = oAbfrageergebnis.getInt(1)
        WEND
        inStart = 1
```

```
    END IF
```

Die Variablen werden deklariert. Die Datenbankverbindung wird, wie weiter oben erklärt, für den Dialog hergestellt. Zum Start ist **loID 0**. Für diesen Fall wird per SQL der niedrigste Wert für den Primärschlüssel ermittelt. Der entsprechende Datensatz soll in dem Dialog später angezeigt werden. Gleichzeitig wird die Variable **inStart** auf 1 gestellt, damit der Dialog später gestartet wird. Enthält die Tabelle keine Daten, so bleibt **loID 0**. Entsprechend muss auch nicht nach dem Inhalt und der Anzahl irgendwelcher Datensätze im Folgenden gesucht werden.

Nur wenn **loID** größer als 0 ist, wird zuerst mit einer Abfrage überprüft, welche Daten in dem Datensatz enthalten sind. Anschließend werden in einer zweiten Abfrage alle Datensätze für die Datensatzanzeige gezählt. Mit der dritten Abfrage wird die Position des aktuellen Datensatzes ermittelt, indem alle Datensätze, die einen kleineren oder eben den aktuellen Primärschlüssel haben, zusammengezählt werden.

```
    IF loID > 0 THEN
        stSql = "SELECT * FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        WHILE oAbfrageergebnis.next
            loID = oAbfrageergebnis.getInt(1)
            stVorname = oAbfrageergebnis.getString(2)
            stNachname = oAbfrageergebnis.getString(3)
        WEND
        stSql = "SELECT COUNT(""ID"") FROM ""Name"""
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        WHILE oAbfrageergebnis.next
            loRowMax = oAbfrageergebnis.getInt(1)
        WEND
        stSql = "SELECT COUNT(""ID"") FROM ""Name"" WHERE ""ID"" <= '"+loID+"'"
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        WHILE oAbfrageergebnis.next
            loRow = oAbfrageergebnis.getInt(1)
        WEND
        oDialog2.getControl("NumericField1").Value = loID
        oDialog2.getControl("TextField1").Text = stVorname
        oDialog2.getControl("TextField2").Text = stNachname
    END IF
    oDialog2.getControl("NumericField2").Value = loRow
    oDialog2.getControl("NumericField3").Value = loRowMax
    IF loRow = 1 THEN
        ' Vorheriger Datensatz
        oDialog2.getControl("CommandButton4").Model.enabled = False
    ELSE
        oDialog2.getControl("CommandButton4").Model.enabled = True
    END IF
    IF loRow <= loRowMax THEN
        ' Nächster Datensatz | Neuer Datensatz | Löschen
        oDialog2.getControl("CommandButton5").Model.enabled = True
        oDialog2.getControl("CommandButton2").Model.enabled = True
        oDialog2.getControl("CommandButton6").Model.enabled = True
    ELSE
        oDialog2.getControl("CommandButton5").Model.enabled = False
        oDialog2.getControl("CommandButton2").Model.enabled = False
        oDialog2.getControl("CommandButton6").Model.enabled = False
    END IF
    IF inStart = 1 THEN
        oDialog2.Execute()
    END IF
END SUB
```

Die ermittelten Werte für die Formularfelder werden übertragen. Die Einträge für die Nummer des aktuellen Datensatzes sowie die Anzahl aller Datensätze werden auf jeden Fall mit einer Zahl versorgt. Ist kein Datensatz vorhanden, so wird hier über den Default-Wert für eine numerische Variable 0 eingefügt.

Die Buttons zum Navigieren > («CommandButton5») und < («CommandButton4») sind nur verfügbar, wenn es möglich ist, einen entsprechenden Datensatz über die Navigation zu erreichen. Ansonsten werden sie vorübergehend mit **`enabled = False`** deaktiviert. Gleiches gilt für die Buttons Neu und Löschen. Sie sollen dann nicht verfügbar sein, wenn die Zahl der angezeigten Zeilen höher ist als die maximal ermittelte Zeilenzahl. Dies ist für die Eingabe neuer Datensätze die Standardeinstellung dieses Dialogs.

Der Dialog soll möglichst nur dann gestartet werden, wenn er direkt aus einer Startdatei über **`DatenLaden(0)`** erstellt werden soll. Deshalb wurde die gesonderte Variable **`inStart`** mit dem Wert 1 zu Beginn der Prozedur versehen..

Über den Button < soll zu dem vorhergehenden Datensatz navigiert werden können. Der Button ist nur dann aktiv ist, wenn nicht bereits der erste Datensatz angezeigt wird. Zum Navigieren wird von dem aktuellen Datensatz der Wert für den Primärschlüssel aus dem Feld «NumericField1» ausgelesen.

Hier gilt es zwei Fälle zu unterscheiden:

1. Es wurde vorher vorwärts zu einer Neueingabe navigiert, so dass das entsprechende Feld keinen Wert enthält. **`loID`** gibt dann den Standardwert wieder, der durch die Definition als Zahlenvariable vorgegeben ist: 0.
2. Ansonsten enthält loID einen Wert, der größer als 0 ist. Entsprechend kann über eine Abfrage die nächstkleinere «ID» ermittelt werden.

```
SUB vorherigerDatensatz
    DIM loID AS LONG
    DIM loIDneu AS LONG
    loID = oDialog2.getControl("NumericField1").Value
    oDatenquelle = thisDatabaseDocument.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    IF loID < 1 THEN
        stSql = "SELECT MAX(""ID"") FROM ""Name"""
    ELSE
        stSql = "SELECT MAX(""ID"") FROM ""Name"" WHERE ""ID"" < '"+loID+"'"
    END IF
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
    WHILE oAbfrageergebnis.next
        loIDneu = oAbfrageergebnis.getInt(1)
    WEND
    IF loIDneu > 0 THEN
        DatenLaden(loIDneu)
    END IF
END SUB
```

Bei einem leeren «ID»-Feld soll auf den Datensatz mit dem höchsten Wert in der Primärschlüsselnummer gewechselt werden. Können hingegen aus dem «ID»-Feld Daten entnommen werden, so wird der entsprechend nachrangige Wert für die "ID" ermittelt.

Das Ergebnis dieser Abfrage dient dazu, die Prozedur «DatenLaden» mit dem entsprechenden Schlüsselwert erneut durchlaufen zu lassen.

Über den Button > wird zum nächsten Datensatz navigiert. Diese Navigationsmöglichkeit steht nur zur Verfügung, wenn nicht bereits der Dialog für die Eingabe eines neuen Datensatzes geleert wurde. Dies ist natürlich auch beim Start und leerer Tabelle der Fall.

Zwangsläufig ist in dem Feld «NumericField1» ein Wert vorhanden. Von diesem Wert ausgehend kann also per SQL nachgesehen werden, welcher Primärschlüsselwert der nächsthöhere in der Tabelle ist. Bleibt die Abfrage leer, weil es keinen entsprechenden Datensatz gibt, so ist der Wert für **`loIDneu = 0`**. Ansonsten kann über die Prozedur «DatenLaden» der Inhalt des nächsten Datensatzes geladen werden.

```
SUB naechsterDatensatz
    DIM loID AS LONG
    DIM loIDneu AS LONG
    loID = oDialog2.getControl("NumericField1").Value
    oDatenquelle = thisDatabaseDocument.CurrentController
    If NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "SELECT MIN(""ID"") FROM ""Name"" WHERE ""ID"" > '"+loID+"'"
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
    WHILE oAbfrageergebnis.next
        loIDneu = oAbfrageergebnis.getInt(1)
    WEND
    IF loIDneu > 0 THEN
        DatenLaden(loIDneu)
    ELSE
        Datenfelder2Leeren
    END IF
END SUB
```

Existiert beim Navigieren zum nächsten Datensatz kein weiterer Datensatz, so löst die Navigation die folgende Prozedur «Datenfelder2Leeren» aus, die zur Eingabe neuer Daten dient.

Mit der Prozedur «Datenfelder2Leeren» werden nicht nur die Datenfelder selbst geleert. Die Position des aktuellen Datensatzes wird um einen Datensatz höher als die maximale Datensatzzahl eingestellt. Das soll verdeutlichen, dass der aktuell bearbeitete Datensatz noch nicht in der Datenbank enthalten ist.

Sobald «Datenfelder2Leeren» ausgelöst wird, wird außerdem die Möglichkeit des Sprungs zum vorhergehenden Datensatz aktiviert. Sprünge zu einem nachfolgenden Datensatz, das erneute Aufrufen der Prozedur über Neu oder das Löschen sind deaktiviert.

```
SUB Datenfelder2Leeren
    loRowMax = oDialog2.getControl("NumericField3").Value
    oDialog2.getControl("NumericField1").Text = ""
    oDialog2.getControl("TextField1").Text = ""
    oDialog2.getControl("TextField2").Text = ""
    oDialog2.getControl("NumericField2").Value = loRowMax + 1
    oDialog2.getControl("CommandButton4").Model.enabled = True  ' Vorheriger Datensatz
    oDialog2.getControl("CommandButton5").Model.enabled = False ' Nächster Datensatz
    oDialog2.getControl("CommandButton2").Model.enabled = False ' Neuer Datensatz
    oDialog2.getControl("CommandButton6").Model.enabled = False ' Löschen
END SUB
```

Das Speichern der Daten soll nur möglich sein, wenn in den Feldern für «ID» und «Nachname» ein Eintrag erfolgt ist. Ist diese Bedingung erfüllt, so wird überprüft, ob der Datensatz ein neuer Datensatz ist. Das funktioniert über den Datensatzanzeiger, der bei neuen Datensätzen so eingestellt wurde, dass er für den aktuellen Datensatz einen um 1 höheren Wert als den maximalen Wert an Datensätzen ausgibt.

Im Falle eines neuen Datensatzes gibt es weiteren Überprüfungsbedarf, damit eine Speicherung einwandfrei funktionieren kann. Kommt die Ziffer für den Primärschlüssel bereits einmal vor, so erfolgt eine Warnung. Wird die entsprechende Frage mit Ja bestätigt, so wird der alte Datensatz mit der gleichen Schlüsselnummer überschrieben. Ansonsten erfolgt keine Speicherung. Solange noch gar kein Datensatz in der Datenbank enthalten ist (**`loRowMax = 0`**) braucht diese Überprüfung nicht zu erfolgen. In dem Falle kann der Datensatz direkt als neuer Datensatz abgespeichert werden. Bei einem neuen Datensatz wird schließlich noch die Zahl der Datensätze um 1 erhöht und die Eingabe für den nächsten Datensatz frei gemacht.

Bei bestehenden Datensätzen wird einfach der alte Datensatz durch ein Update mit dem neuen Datensatz überschrieben.

```
SUB Daten2Speichern(oEvent AS OBJECT)
    DIM oDatenquelle AS OBJECT
```

```
    DIM oVerbindung AS OBJECT
    DIM oSQL_Anweisung AS OBJECT
    DIM oDlg AS OBJECT
    DIM loID AS LONG
    DIM stVorname AS STRING
    DIM stNachname AS STRING
    DIM inMsg AS INTEGER
    DIM loRow AS LONG
    DIM loRowMax AS LONG
    DIM stSql AS STRING
    oDlg = oEvent.Source.getContext()
    loID = oDlg.getControl("NumericField1").Value
    stVorname = oDlg.getControl("TextField1").Text
    stNachname = oDlg.getControl("TextField2").Text
    IF loID > 0 AND stNachname <> "" THEN
        oDatenquelle = thisDatabaseDocument.CurrentController
        If NOT (oDatenquelle.isConnected()) THEN
            oDatenquelle.connect()
        END IF
        oVerbindung = oDatenquelle.ActiveConnection()
        oSQL_Anweisung = oVerbindung.createStatement()
        loRow = oDlg.getControl("NumericField2").Value
        loRowMax = oDlg.getControl("NumericField3").Value
        IF loRowMax < loRow THEN
            IF loRowMax > 0 THEN
                stSql = "SELECT ""ID"" FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
                oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
                WHILE oAbfrageergebnis.next
                    inMsg = MsgBox ("Der Wert für das Feld 'ID' existiert schon." &
                        CHR(13) & "Soll der Datensatz überschrieben werden?",20,
                        "Doppelte Dateneingabe")
                    IF inMsg = 6 THEN
                        stSql = "UPDATE ""Name"" SET ""Vorname""='"+stVorname+"',
                            ""Nachname""='"+stNachname+"' WHERE ""ID"" = '"+loID+"'"
                        oSQL_Anweisung.executeUpdate(stSql)
                        DatenLaden(loID) ' Beim Update wurde ein bestehender Datensatz
                            überschrieben. Neueinlasen zur Korrektur der Datensatzzahlen
                    END IF
                    EXIT SUB
                WEND
            END IF
            stSql = "INSERT INTO ""Name"" (""ID"", ""Vorname"", ""Nachname"") VALUES
                ('"+loID+"','"+stVorname+"','"+stNachname+"')"
            oSQL_Anweisung.executeUpdate(stSql)
            oDlg.getControl("NumericField3").Value = loRowMax + 1
                ' Nach dem Insert existiert ein Datensatz mehr
            Datenfelder2Leeren
                ' Nach einem Insert wird grundsätzlich zum nächsten Insert geschaltet
        ELSE
            stSql = "UPDATE ""Name"" SET ""Vorname""='"+stVorname+"',
                ""Nachname""='"+stNachname+"' WHERE ""ID"" = '"+loID+"'"
            oSQL_Anweisung.executeUpdate(stSql)
        END IF
    END IF
END SUB
```

Die Löschprozedur ist mit einer Nachfrage versehen, die versehentliches Löschen verhindern soll. Dadurch, dass der Button deaktiviert wird, wenn die Eingabefelder leer sind, dürfte es nicht vorkommen, dass das Feld «NumericField1» leer ist. Deshalb könnte die Überprüfung der Bedingung **`IF loID > 0`** auch entfallen.

Beim Löschen wird die Zahl der Datensätze um einen Datensatz herabgesetzt. Dies muss entsprechend mit **`loRowMax – 1`** korrigiert werden. Anschließend wird der dem aktuellen Datensatz folgende Datensatz angezeigt.

```
SUB DatenLoeschen(oEvent AS OBJECT)
    DIM oDatenquelle AS OBJECT
    DIM oVerbindung AS OBJECT
```

```
    DIM oSQL_Anweisung AS OBJECT
    DIM oDlg AS OBJECT
    DIM loID AS LONG
    oDlg = oEvent.Source.getContext()
    loID = oDlg.getControl("NumericField1").Value
    IF loID > 0 THEN
       inMsg = MsgBox ("Soll der Datensatz wirklich gelöscht werden?",20,
          "Löschen eines Datensatzes")
       IF inMsg = 6 THEN
          oDatenquelle = thisDatabaseDocument.CurrentController
          If NOT (oDatenquelle.isConnected()) THEN
             oDatenquelle.connect()
          END IF
          oVerbindung = oDatenquelle.ActiveConnection()
          oSQL_Anweisung = oVerbindung.createStatement()
          stSql = "DELETE FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
          oSQL_Anweisung.executeUpdate(stSql)
          loRowMax = oDlg.getControl("NumericField3").Value
          oDlg.getControl("NumericField3").Value = loRowMax - 1
          naechsterDatensatz
       END IF
    ELSE
       MsgBox ("Kein Datensatz gelöscht." & CHR(13) &
          "Es fehlt eine Datensatzauswahl.",64,"Löschung nicht möglich")
    END IF
END SUB
```

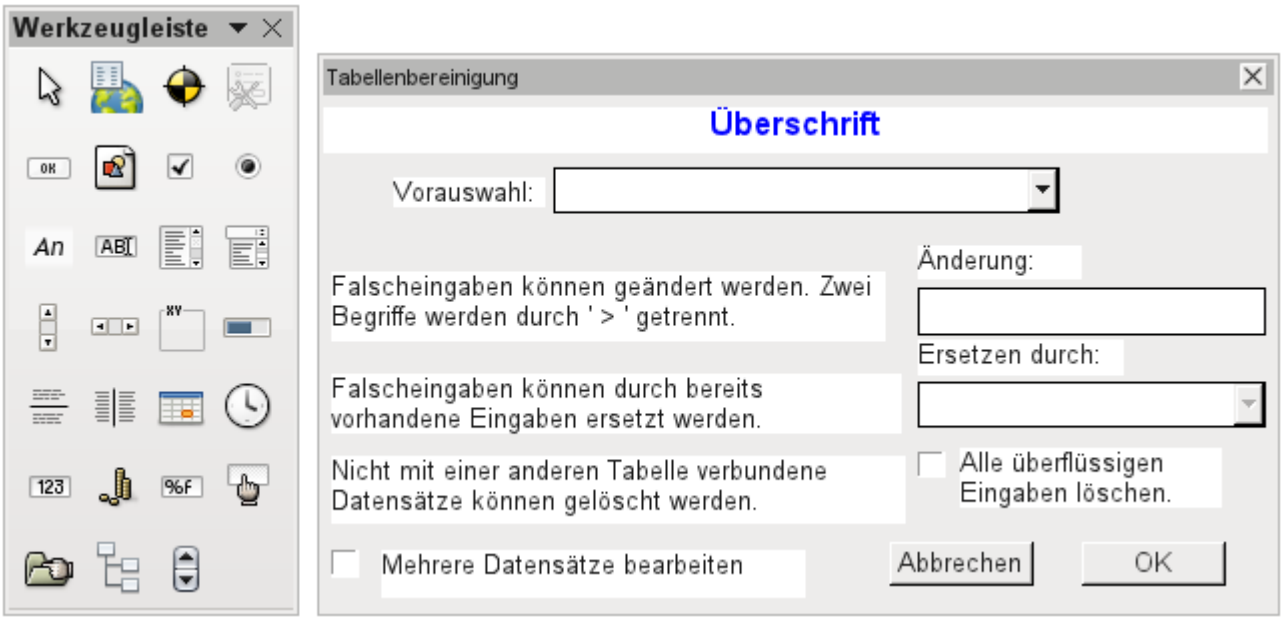
Bereits dieser kleine Dialog zur Bearbeitung von Daten zeigt, dass der Aufwand im Makrocode schon erheblich ist, um die Grundlagen einer Datenbearbeitung zu gewährleisten. Der Zugriff über ein Formular ist hier erheblich einfacher. Der Dialog kann dagegen recht flexibel an die Bedürfnisse des Programms angepasst werden. Nur ist das eben nicht für die Erstellung einer Datenbankbedienung im Schnellverfahren gedacht.

Fehleinträge von Tabellen mit Hilfe eines Dialogs bereinigen

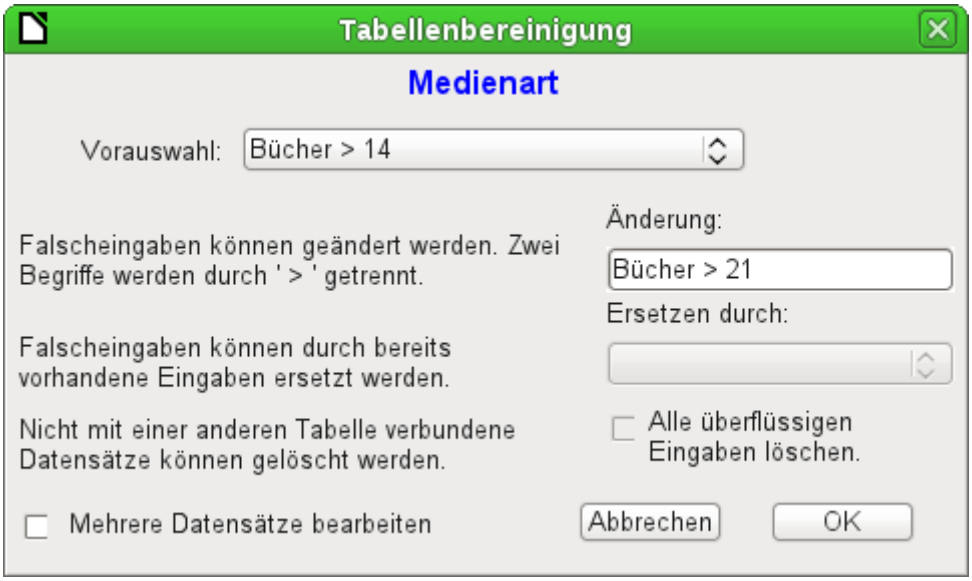
Fehleingaben in Feldern fallen häufig erst später auf. Manchmal müssen auch gleich mehrere Datensätze mit der gleichen Eingabe auf einmal geändert werden. Dies ist in der normalen Tabellenansicht umständlich, je mehr Änderungen vorgenommen werden müssen, da für jeden Datensatz einzeln eine Eingabe erforderlich ist.

Formulare könnten hier mit Makros greifen. Wird aber für viele Tabellen ein identisch aufgebautes Formular benötigt, so bietet sich an, dies mit Dialogen zu erledigen. Ein Dialog wird zu Beginn mit den notwendigen Daten zu der jeweiligen Tabelle versehen und kann so statt mehrerer Formulare genutzt werden.

Diese Dialoge müssen für Firebird wegen der unterschiedlichen Systemtabellen entsprechend angepasst werden. Zu den Systemtabellen siehe den Anhang dieses Handbuches.



Dialoge werden neben den Modulen für Makros abgespeichert. Ihre Erstellung erfolgt ähnlich der eines Formulars. Hier stehen auch weitgehend ähnliche Kontrollfelder zur Verfügung. Lediglich das Tabellenkontrollfeld aus dem Formular fehlt als besondere Eingabemöglichkeit.



Wird ein Dialog ausgeführt, so erscheinen die Kontrollfelder entsprechend der Einstellung der grafischen Benutzeroberfläche.

Der oben abgebildete Dialog der Beispieldatenbank soll dazu dienen, die Tabellen zu bearbeiten, die nicht direkt in einem der Formulare als Grundlage vorhanden sind. So ist z.B. die Medienart über ein Listenfeld zugänglich, in der Makro-Version bereits durch ein Kombinationsfeld. In der Makro-Version können die Inhalte der Felder zwar durch neue Inhalte ergänzt werden, eine Änderung alter Inhalte ist aber nicht möglich. In der Version ohne Makros erfolgt die Änderung über ein separates Tabellenkontrollfeld.

Während die Änderung noch ohne Makros recht einfach in den Griff zu bekommen ist, so ist es doch recht umständlich, die Medienart vieler Medien auf eine andere Medienart zu ändern. Angenommen, es gäbe die Medienarten 'Buch, gebunden', 'Buch, kartoniert', 'Taschenbuch' und 'Ringbuch'. Jetzt stellt sich nach längerem Betrieb der Datenbank heraus, dass muntere Zeitgenossen noch weitere ähnliche Medienarten für gedruckte Werke vorgesehen haben. Nur ist uns die Differenzierung viel zu weitgehend. Es soll also reduziert werden, am liebsten auf nur einen Begriff.

Ohne Makro müssten jetzt die Datensätze in der Tabelle Medien (mit Hilfe von Filtern) aufgesucht werden und einzeln geändert werden. Mit Kenntnis von SQL geht dies über die SQL-Eingabe schon wesentlich besser. Mit einer Eingabe werden alle Datensätze der Tabelle Medien geändert. Mit einer zweiten SQL-Anweisung wird dann die jetzt überflüssige Medienart gelöscht, die keine Verbindung mehr zur Tabelle "Medien" hat. Genau dieses Verfahren wird mit diesem Dialog über «Ersetzen durch:» angewandt – nur dass eben die SQL-Anweisung erst über das Makro an die Tabelle "Medienart" angepasst wird, da das Makro auch andere Tabellen bearbeiten können soll.

Manchmal schleichen sich auch Eingaben in eine Tabelle ein, die im Nachhinein in den Formularen geändert wurden, also eigentlich gar nicht mehr benötigt werden. Da kann es nicht schaden, solche verwaisten Datensätze einfach zu löschen. Nur sind die über die grafische Oberfläche recht schwer ausfindig zu machen. Hier hilft wieder eine entsprechende SQL-Abfrage, die mit einer Löschanweisung gekoppelt ist. Diese Anweisung ist im Dialog je nach betroffener Tabelle unter «Alle überflüssigen Eingaben löschen» hinterlegt.

Sollen mit dem Dialog mehrere Änderungen durchgeführt werden, so ist dies über das Markierfeld «Mehrere Datensätze bearbeiten» anzugeben. Dann endet der Dialog nicht mit der Betätigung des Buttons «OK».

Der Makrocode für diesen Dialog ist aus der Beispieldatenbank ersichtlich. Im Folgenden werden nur Ausschnitte daraus erläutert.

```
SUB Tabellenbereinigung(oEvent AS OBJECT)
```

Das Makro soll über Einträge im Bereich «*Zusatzinformationen*» des jeweiligen Buttons gestartet werden.

```
0: Formular, 1: Unterformular, 2: UnterUnterformular, 3: Kombinationsfeld oder
Tabellenkontrollfeld, 4: Fremdschlüsselfeld im Formular, bei Tabellenkontrollfeld
leer, 5: Tabellenname Nebentabelle, 6: Tabellenfeld1 Nebentabelle, 7: Tabellenfeld2
Nebentabelle, ggf. 8: Tabellenname Nebentabelle für Tabellenfeld 2
```

Die Einträge in diesem Bereich werden zu Beginn des Makros als Kommentar aufgelistet. Die damit verbundenen Ziffern geben die Ziffern wieder, unter denen der jeweilige Eintrag aus dem Array ausgelesen wird. Das Makro kann Listenfelder verarbeiten, die zwei Einträge, getrennt durch «**>**», enthalten. Diese beiden Einträge können auch aus unterschiedlichen Tabellen stammen und über eine Abfrage zusammengeführt sein, wie z.B. bei der Tabelle "Postleitzahl", die für die Orte lediglich das Fremdschlüsselfeld *"*Ort_ID*"* enthält, zur Darstellung des Ortes also die Tabelle "Ort" benötigt.

```
DIM aFremdTabellen(0, 0 to 1)
DIM aFremdTabellen2(0, 0 to 1)
```

Unter den zu Beginn definierten Variablen fallen zwei Arrays auf. Während normale Arrays auch durch den Befehl **'Split()'** während der Laufzeit der Prozedur erstellt werden können, müssen zweidimensionale Arrays vorher definiert werden. Zweidimensionale Arrays werden z.B. benötigt, um aus einer Abfrage mehrere Datensätze zu speichern, bei denen die Abfrage selbst über mehr als ein Feld geht. Die beiden obigen Arrays müssen Abfragen auswerten, die sich jeweils auf zwei Tabellenfelder beziehen. Deshalb werden sie in der zweiten Dimension mit *'0 to 1'* auf zwei unterschiedliche Inhalte festgelegt.

```
stTag = oEvent.Source.Model.Tag
aTabelle() = Split(stTag, ", ")
FOR i = LBound(aTabelle()) TO UBound(aTabelle())
   aTabelle(i) = trim(aTabelle(i))
NEXT
```

Die mitgegebenen Variablen werden ausgelesen. Die Reihenfolge steht im obigen Kommentar. Es gibt maximal 9 Einträge, wobei geklärt werden muss, ob ein 8. Eintrag für das Tabellenfeld2 und ein 9. Eintrag für eine zweite Tabelle existieren.

Wenn Werte aus einer Tabelle entfernt werden, so muss zuerst einmal berücksichtigt werden, ob sie nicht noch als Fremdschlüssel in anderen Tabellen existieren. In einfachen Tabellenkonstruktionen gibt es von einer Tabelle aus lediglich eine Fremdschlüsselverbindung zu einer anderen

Tabelle. In der vorliegenden Beispieldatenbank aber wird z.B. die Tabelle "Ort" genutzt, um die Erscheinungsorte der Medien und die Orte für die Adressen zu speichern. Es wird also zweimal der Primärschlüssel der Tabelle "Ort" in unterschiedlichen Tabellen eingetragen. Diese Tabellen und Fremdschlüsselbezeichnungen könnten natürlich auch über die «Zusatzinformationen» eingegeben werden. Schöner wäre es aber, wenn sie universell für alle Fälle ermittelt werden. Dies geschieht durch die folgende Abfrage.

```
stSql = "SELECT ""FKTABLE_NAME"", ""FKCOLUMN_NAME"" FROM
    ""INFORMATION_SCHEMA"".""SYSTEM_CROSSREFERENCE"" WHERE ""PKTABLE_NAME"" = '"
    + aTabelle(5) + "'"
```

In der Datenbank sind im Bereich "INFORMATION_SCHEMA" alle Informationen zu den Tabellen der Datenbank abgespeichert, so auch die Informationen zu den Fremdschlüsseln. Die entsprechende Tabelle, die diese Informationen enthält, ist über "INFORMATION_SCHEMA"."SYSTEM_CROSSREFERENCE" erreichbar. Mit "PKTABLE_NAME" wird die Tabelle erreicht, die ihren Primärschlüssel ("Primary Key") in die Beziehung mit einbringt. Mit "FKTABLE_NAME" wird die Tabelle erreicht, die diesen Primärschlüssel als Fremdschlüssel ("Foreign Key") nutzt. Über "FKCOLUMN_NAME" wird schließlich die Bezeichnung des Fremdschlüsselfeldes ermittelt.

Die Tabelle, die einen Primärschlüssel als Fremdschlüssel zur Verfügung stellt, befindet sich in dem vorher erstellten Array an der 6. Position. Da die Zählung mit 0 beginnt, wird der Wert aus dem Array mit **`aTabelle(5)`** ermittelt.

```
inZaehler = 0
stFremdIDTab1Tab2 = "ID"
stFremdIDTab2Tab1 = "ID"
stNebentabelle = aTabelle(5)
```

Bevor die Auslesung des Arrays gestartet wird, müssen einige Standardwerte gesetzt werden. Dies sind der Zähler für das Array, in das die Werte der Nebentabelle geschrieben werden, der Standardprimärschlüssel, wenn nicht der Fremdschlüssel für eine zweite Tabelle benötigt wird und die Standardnebentabelle, die sich auf die Haupttabelle bezieht, bei Postleitzahl und Ort z.B. die Tabelle für die Postleitzahl.

Bei der Verknüpfung von zwei Feldern zur Anzeige in den Listenfeldern kann es ja, wie oben erwähnt, zu einer Verknüpfung über zwei Tabellen kommen. Für die Darstellung von Postleitzahl und Ort lautet hier die Abfrage

```
SELECT "Postleitzahl"."Postleitzahl" || ' > ' || "Ort"."Ort"
FROM "Postleitzahl", "Ort"
WHERE "Postleitzahl"."Ort_ID" = "Ort"."ID"
```

Die Tabelle, die sich auf das erste Feld bezieht (Postleitzahl), ist mit der zweiten Tabelle über einen Fremdschlüssel verbunden. Lediglich die Information der beiden Tabellen und der Felder "Postleitzahl" und "Ort" wurde dem Makro mitgegeben. Die Primärschlüssel sind standardmäßig in dem Beispiel mit der Bezeichnung "ID" versehen. Der Fremdschlüssel von ″Ort″ in ″Postleitzahl″ muss also über das Makro ermittelt werden.

Genauso muss über das Makro jede andere Tabelle ermittelt werden, mit der die Inhalte des Listenfeldes über Fremdschlüssel in Verbindung stehen.

```
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
WHILE oAbfrageergebnis.next
    ReDim Preserve aFremdTabellen(inZaehler,0 to 1)
```

Das Array muss jedes Mal neu dimensioniert werden. Damit die alten Inhalte erhalten bleiben, erfolgt über (Preserve) eine Sicherung des vorherigen Inhaltes.

```
    aFremdTabellen(inZaehler,0) = oAbfrageergebnis.getString(1)
```

Auslesen des ersten Feldes mit dem Namen der Tabelle, die den Fremdschlüssel enthält. Ergebnis für die Tabelle ″Postleitzahl″ ist hier die Tabelle "Adresse".

```
    aFremdTabellen(inZaehler,1) = oAbfrageergebnis.getString(2)
```

Auslesen des zweiten Feldes mit der Bezeichnung des Fremdschlüsselfeldes. Ergebnis für die Tabelle "Postleitzahl" ist hier das Feld "Postleitzahl_ID" in der Tabelle "Adresse".

Für den Fall, dass dem Aufruf der Prozedur auch der Name einer zweiten Tabelle mitgegeben wurde, erfolgt die folgende Schleife. Nur wenn der Name der zweiten Tabelle als Fremdschlüsseltabelle für die erste Tabelle auftaucht, erfolgt hier eine Änderung der Standardeinträge. In unserem Fall kommt dies nicht vor, da die Tabelle "Ort" keinen Fremdschlüssel der Tabelle "Postleitzahl" enthält. Der Standardeintrag für die Nebentabelle bleibt also bei "Postleitzahl"; schließlich ist die Kombination von Postleitzahl und Ort eine Grundlage für die Adressentabelle, die einen Fremdschlüssel zu der Tabelle "Postleitzahl" enthält.

```
IF UBound(aTabelle()) = 8 THEN
    IF aTabelle(8) = aFremdTabellen(inZaehler,0) THEN
        stFremdIDTab2Tab1 = aFremdTabellen(inZaehler,1)
        stNebentabelle = aTabelle(8)
    END IF
END IF
inZaehler = inZaehler + 1
```

Da eventuell noch weitere Werte auszulesen sind, erfolgt eine Erweiterung des Zählers zur Neudimensionierung des Arrays. Anschließend wird die Schleife beendet.

```
WEND
```

Existiert im Aufruf der Prozedur ein zweiter Tabellenname, so wird die gleiche Abfrage jetzt mit dem zweiten Tabellennamen gestartet:

```
IF UBound(aTabelle()) = 8 THEN
```

Der Ablauf ist identisch. Nur wird in der Schleife jetzt gesucht, ob vielleicht der erste Tabellenname als Fremdschlüssel-Tabellenname auftaucht. Das ist hier der Fall: Die Tabelle "Postleitzahl" enthält den Fremdschlüssel "Ort_ID" aus der Tabelle "Ort". Dieser Fremdschlüssel wird also jetzt der Variablen «stFremdIDTab1Tab2» zugewiesen, so dass die Beziehung der Tabellen untereinander definiert werden kann.

```
            IF aTabelle(5) = aFremdTabellen2(inZaehler,0) THEN
                stFremdIDTab1Tab2 = aFremdTabellen2(inZaehler,1)
            END IF
```

Nach einigen weiteren Einstellungen zur korrekten Rückkehr nach Aufruf des Dialogs in die entsprechenden Formulare (Ermittlung der Zeilennummer des Formulars, damit nach einem Neueinlesen auf die Zeilennummer wieder gesprungen werden kann) startet die Schleife, die den Dialog gegebenenfalls wieder neu erstellt, wenn die erste Aktion erfolgt ist, der Dialog aber für weitere Aktionen offen gehalten werden soll. Die Einstellung zur Wiederholung erfolgt über das entsprechende Markierfeld.

```
DO
```

Bevor der Dialog gestartet wird, wird erst einmal der Inhalt der Listenfelder ermittelt. Dabei muss berücksichtigt werden, ob die Listenfelder zwei Tabellenfelder darstellen und eventuell sogar einen Bezug zu zwei Tabellen haben.

```
IF UBound(aTabelle()) = 6 THEN
```

Das Listenfeld bezieht sich nur auf eine Tabelle und ein Feld, da das Array bei dem Tabellenfeld1 der Nebentabelle endet.

```
    stSql = "SELECT """ + aTabelle(6) + """ FROM """ + aTabelle(5)
        + """ ORDER BY """ + aTabelle(6) + """"
ELSEIF UBound(aTabelle()) = 7 THEN
```

Das Listenfeld bezieht sich auf zwei Tabellenfelder, aber nur auf eine Tabelle, da das Array bei dem Tabellenfeld2 der Nebentabelle endet.

```
    stSql = "SELECT """ + aTabelle(6) + """||' > '||""" + aTabelle(7)
        + """ FROM """ + aTabelle(5) + """ ORDER BY """ + aTabelle(6) + """"
ELSE
```

Das Listenfeld hat zwei Tabellenfelder und zwei Tabellen als Grundlage. Diese Abfrage trifft also auf das Beispiel mit der Postleitzahl und den Ort zu.

```
            stSql = "SELECT """ + aTabelle(5) + """."""" + aTabelle(6) + """||' > '||"""
                  + aTabelle(8) + """."""" + aTabelle(7) + """ FROM """ + aTabelle(5)
                  + """, """ + aTabelle(8) + """ WHERE """ + aTabelle(8) + """."""
                  + stFremdIDTab2Tab1 + """ = """ + aTabelle(5) + """."""
                  + stFremdIDTab1Tab2 + """ ORDER BY """ + aTabelle(6) + """"
        END IF
```

Hier erfolgt die erste Auswertung zur Ermittlung von Fremdschlüsseln. Die Variablen «stFremdIDTab2Tab1» und «stFremdIDTab1Tab2» starten mit dem Wert *"ID"*. Für «stFremdIDTab1Tab2» wurde in der Auswertung der vorhergehenden Abfrage ein anderer Wert ermittelt, nämlich der Wert *"Ort_ID"*. Damit ergibt die vorherige Abfragekonstruktion genau den Inhalt, der weiter oben bereits für Postleitzahl und Ort formuliert wurde – lediglich erweitert um die Sortierung.

Jetzt muss der Kontakt zu den Listenfeldern erstellt werden, damit diese mit dem Inhalt der Abfragen bestückt werden. Diese Listenfelder existieren noch nicht, da noch gar kein Dialog existiert. Dieser Dialog wird mit den folgenden Zeilen erst einmal im Speicher erstellt, bevor er tatsächlich auf dem Bildschirm ausgeführt wird.

```
        DialogLibraries.LoadLibrary("Standard")
        oDlg = CreateUnoDialog(DialogLibraries.Standard.Dialog_Tabellenbereinigung)
```

Anschließend werden Einstellungen für die Felder, die der Dialog enthält, ausgeführt. Hier als Beispiel das Auswahllistenfeld, das mit dem Ergebnis der obigen Abfrage bestückt wird:

```
        oCtlList1 = oDlg.GetControl("ListBox1")
        oCtlList1.addItems(aInhalt(),0)
```

Der Zugriff auf die Felder des Dialogs erfolgt über **`GetControl`** sowie die entsprechende Bezeichnung. Bei Dialogen ist es nicht möglich, für zwei Felder die gleichen Bezeichnungen zu verwenden, da sonst eine Auswertung des Dialoges problematisch wäre.

Das Listenfeld wird mit den Inhalten aus der Abfrage, die in dem Array «aInhalt()» gespeichert wurden, ausgestattet. Das Listenfeld enthält nur die darzustellenden Inhalte als ein Feld, wird also nur in der Position '0' bestückt.

Nachdem alle Felder mit den gewünschten Inhalten versorgt wurden, wird der Dialog gestartet.

```
        Select Case oDlg.Execute()
        Case 1 'Case 1 bedeutet die Betätigung des Buttons "OK"
        Case 0 'Wenn Button "Abbrechen"
            inWiederholung = 0
        End Select
    LOOP WHILE inWiederholung = 1
```

Der Dialog wird so lange durchgeführt, wie der Wert für «inWiederholung» auf 1 steht. Diese Setzung erfolgt mit dem entsprechenden Markierfeld.

Hier der Inhalt nach Betätigung des Buttons «OK» im Kurzüberblick:

```
    Case 1
        stInhalt1 = oCtlList1.getSelectedItem() 'Wert aus Listbox1 auslesen ...
        REM ... und den dazugehoerigen ID-Wert bestimmen.
```

Der ID-Wert des ersten Listenfeldes wird in der Variablen «inLB1» gespeichert.

```
        stText = oCtlText.Text  ' Den Wert des Feldes auslesen.
```

Ist das Textfeld nicht leer, so wird nur der Eintrag im Textfeld erledigt. Weder das Listenfeld für eine andere Zuweisung noch das Markierfeld für eine Löschung aller Daten ohne Bezug werden berücksichtigt. Dies wird auch dadurch verdeutlicht, dass bei Texteingabe die anderen Felder inaktiv geschaltet werden.

```
        IF stText <> "" THEN
```

Ist das Textfeld nicht leer, dann wird der neue Wert anstelle des alten Wertes mit Hilfe des vorher ausgelesenen ID-Feldes in die Tabelle geschrieben. Dabei werden wieder zwei Einträge ermöglicht, wie dies auch in dem Listenfeld geschieht. Das Trennzeichen ist «**>**». Bei Zwei Einträgen in verschiedenen Tabellen müssen auch entsprechend zwei UPDATE-Kommandos gestartet werden, die hier gleichzeitig erstellt und, durch ein Semikolon getrennt, weitergeleitet werden.

```
        ELSEIF oCtlList2.getSelectedItem() <> "" THEN
```

Wenn das Textfeld leer ist und das Listenfeld 2 einen Wert aufweist, muss der Wert des Listenfeldes 1 durch den Wert des Listenfeldes 2 ersetzt werden. Das bedeutet, dass alle Datensätze der Tabellen, in denen die Datensätze der Listenfelder Fremdschlüssel sind, überprüft und gegebenenfalls mit einem geänderten Fremdschlüssel beschrieben werden müssen.

```
        stInhalt2 = oCtlList2.getSelectedItem()
        REM Den Wert der Listbox auslesen.
        REM ID für den Wert das Listenfeld ermitteln.
```

Der ID-Wert des zweiten Listenfeldes wird in der Variablen «inLB2» gespeichert. Auch dieses erfolgt wieder unterschiedlich, je nachdem, ob ein oder zwei Felder in dem Listenfeld enthalten sind sowie eine oder zwei Tabellen Ursprungstabellen des Listenfeldinhaltes sind.

Der Ersetzungsprozess erfolgt danach, welche Tabelle als die Tabelle definiert wurde, die für die Haupttabelle den Fremdschlüssel darstellt. Für das oben erwähnte Beispiel ist dies die Tabelle "Postleitzahl", da die "Postleitzahl_ID" der Fremdschlüssel ist, der durch Listenfeld 1 und Listenfeld 2 wiedergegeben wird.

```
    IF stNebentabelle = aTabelle(5) THEN
        FOR i = LBound(aFremdTabellen()) TO UBound(aFremdTabellen())
```

Ersetzen des alten ID-Wertes durch den neuen ID-Wert. Problematisch ist dies bei n:m-Beziehungen, da dann der gleiche Wert doppelt zugeordnet werden kann. Dies kann erwünscht sein, muss aber vermieden werden, wenn der Fremdschlüssel hier Teil des Primärschlüssels ist. So darf in der Tabelle "rel_Medien_Verfasser" ein Medium nicht zweimal den gleichen Verfasser haben, da der Primärschlüssel aus der "Medien_ID" und der "Verfasser_ID" gebildet wird. In der Abfrage werden alle Schlüsselfelder untersucht, die zusammen die Eigenschaft UNIQUE haben oder als Fremdschlüssel mit der Eigenschaft 'UNIQUE' über einen Index definiert wurden.

Sollte also der Fremdschlüssel die Eigenschaft 'UNIQUE' haben und bereits mit der gewünschten zukünftigen «inLB2» dort vertreten sein, so kann der Schlüssel nicht ersetzt werden.

```
stSql = "SELECT ""COLUMN_NAME"" FROM ""INFORMATION_SCHEMA"".""SYSTEM_INDEXINFO""
    WHERE ""TABLE_NAME"" = '" + aFremdTabellen(i,0) + "' AND ""NON_UNIQUE"" = False
    AND ""INDEX_NAME"" = (SELECT ""INDEX_NAME"" FROM
    ""INFORMATION_SCHEMA"".""SYSTEM_INDEXINFO"" WHERE ""TABLE_NAME"" = '"
    + aFremdTabellen(i,0) + "' AND ""COLUMN_NAME"" = '" + aFremdTabellen(i,1) + "')"
```

Mit **`"NON_UNIQUE" = False`** werden die Spaltennamen angegeben, die 'UNIQUE' sind. Allerdings werden nicht alle Spaltennamen benötigt, sondern nur die, die gemeinsam mit dem Fremdschlüsselfeld einen Index bilden. Dies ermittelt der 'Subselect' mit dem gleichen Tabellennamen (der den Fremdschlüssel enthält) und dem Namen des Fremdschlüsselfeldes.

Wenn jetzt der Fremdschlüssel in der Ergebnismenge vorhanden ist, dann darf der Schlüsselwert nur dann ersetzt werden, wenn gleichzeitig andere Felder dazu benutzt werden, den entsprechenden Index als 'UNIQUE' zu definieren. Hierzu muss beim Ersetzen darauf geachtet werden, dass die Einzigartigkeit der Indexkombination nicht verletzt wird.

```
    IF aFremdTabellen(i,1) = stFeldbezeichnung THEN
        inUnique = 1
    ELSE
        ReDim Preserve aSpalten(inZaehler)
        aSpalten(inZaehler) = oAbfrageergebnis.getString(1)
        inZaehler = inZaehler + 1
    END IF
```

Alle Spaltennamen, die neben dem bereits bekannten Spaltennamen des Fremdschlüsselfeldes als Index mit der Eigenschaft UNIQUE auftauchen, werden in einem Array abgespeichert. Da der Spaltenname des Fremdschlüsselfeldes auch zu der Gruppe gehört, wird durch ihn gekennzeichnet, dass die Einzigartigkeit bei der Datenänderung zu berücksichtigen ist.

```
IF inUnique = 1 THEN
    stSql = "UPDATE """ + aFremdTabellen(i,0) + """ AS ""a"" SET """
        + aFremdTabellen(i,1) + """='" + inLB2 + "' WHERE """ + aFremdTabellen(i,1)
```

```
        + """='" + inLB1 + "' AND ( SELECT COUNT(*) FROM """ + aFremdTabellen(i,0)
        + """ WHERE """ + aFremdTabellen(i,1) + """='" + inLB2 + "' )"
    IF inZaehler > 0 THEN
        stFeldgruppe = Join(aSpalten(), """||  ||""")
```

Gibt es mehrere Felder, die neben dem Fremdschlüsselfeld gemeinsam einen 'UNIQUE'-Index bilden, so werden die hier für eine SQL-Gruppierung zusammengeführt. Ansonsten erscheint als «stFeldgruppe» nur «aSpalten(0)».

```
        stFeldbezeichnung = ""
        FOR ink = LBound(aSpalten()) TO UBound(aSpalten())
            stFeldbezeichnung = stFeldbezeichnung + " AND """ + aSpalten(ink)
                + """ = ""a"".""" + aSpalten(ink) + """ "
```

Die SQL-Teilstücke werden für eine korrelierte Unterabfrage zusammengefügt.

```
        NEXT
    stSql = Left(stSql, Len(stSql) - 1)
```

Die vorher erstellte Abfrage endet mit einer Klammer. Jetzt sollen noch Inhalte zu der Unterabfrage hinzugefügt werden. Also muss die Schließung wieder aufgehoben werden. Anschließend wird die Abfrage durch die zusätzlich ermittelten Bedingungen ergänzt.

```
        stSql = stSql + stFeldbezeichnung + "GROUP BY (""" + stFeldgruppe + """) ) < 1"
END IF
```

Wenn die Feldbezeichnung des Fremdschlüssels nichts mit dem Primärschlüssel oder einem 'UNIQUE'-Index zu tun hat, dann kann ohne weiteres auch ein Inhalt doppelt erscheinen

```
ELSE
    stSql = "UPDATE """ + aFremdTabellen(i,0) + """ SET """ + aFremdTabellen(i,1)
        + """='" + inLB2 + "' WHERE """ + aFremdTabellen(i,1) + """='" + inLB1 + "'"
END IF
oSQL_Anweisung.executeQuery(stSql)
NEXT
```

Das Update wird so lange durchgeführt, wie unterschiedliche Verbindungen zu anderen Tabellen vorkommen, d. h. die aktuelle Tabelle einen Fremdschlüssel in anderen Tabellen liegen hat. Dies ist z. B. bei der Tabelle "Ort" zweimal der Fall: in der Tabelle "Medien" und in der Tabelle "Postleitzahl".

Anschließend kann der alte Wert aus dem Listenfeld 1 gelöscht werden, weil er keine Verbindung mehr zu anderen Tabellen hat.

```
stSql = "DELETE FROM """ + aTabelle(5) + """ WHERE ""ID""='" + inLB1 + "'"
oSQL_Anweisung.executeQuery(stSql)
```

Das gleiche Verfahren muss jetzt auch für eine eventuelle zweite Tabelle durchgeführt werden, aus der die Listenfelder gespeist werden. In unserem Beispiel ist die erste Tabelle die Tabelle "Postleitzahl", die zweite Tabelle die Tabelle "Ort".

Wenn das Textfeld leer ist und das Listenfeld 2 ebenfalls nichts enthält, wird nachgesehen, ob eventuell das Markierfeld darauf hindeutet, dass alle überflüssigen Einträge zu löschen sind. Dies ist für die Einträge der Fall, die nicht mit anderen Tabellen über einen Fremdschlüssel verbunden sind.

```
ELSEIF oCtlCheck1.State = 1 THEN
    stBedingung = ""
    IF stNebentabelle = aTabelle(5) THEN
        FOR i = LBound(aFremdTabellen()) TO UBound(aFremdTabellen())
            stBedingung = stBedingung + """ID"" NOT IN (SELECT """
                + aFremdTabellen(i,1) + """ FROM """ + aFremdTabellen(i,0) + """) AND "
        NEXT
    ELSE
        FOR i = LBound(aFremdTabellen2()) TO UBound(aFremdTabellen2())
            stBedingung = stBedingung + """ID"" NOT IN (SELECT """ +
                aFremdTabellen2(i,1) + """ FROM """ + aFremdTabellen2(i,0) + """) AND "
        NEXT
    END IF
```

Das letzte «AND» muss abgeschnitten werden, da sonst die Löschanweisung mit einem «AND» enden würde.

```
        stBedingung = Left(stBedingung, Len(stBedingung) - 4)'
        stSql = "DELETE FROM """ + stNebentabelle + """ WHERE " + stBedingung + ""
        oSQL_Anweisung.executeQuery(stSql)
```

Da nun schon einmal die Tabelle bereinigt wurde, kann auch gleich der Tabellenindex überprüft und gegebenenfalls nach unten korrigiert werden. Siehe hierzu die in dem vorhergehenden Kapitel *Tabellenindex heruntersetzen bei Autowert-Feldern* erwähnte Prozedur.

```
    Tabellenindex_runter(stNebentabelle)
```

Anschließend wird noch gegebenenfalls das Listenfeld des Formulars, aus dem der Tabellenbereinigungsdialog aufgerufen wurde, auf den neuesten Stand gebracht. Unter Umständen ist das gesamte Formular neu einzulesen. Hierzu wurde zu Beginn der Prozedur der aktuelle Datensatz ermittelt, so dass nach einem Auffrischen des Formulars der aktuelle Datensatz auch wieder eingestellt werden kann.

```
        oDlg.endExecute()    'Dialog beenden ...
        oDlg.Dispose()   '... und aus dem Speicher entfernen
    END SUB
```

Dialoge werden mit **`endExecute()`** beendet und mit **`Dispose()`** komplett aus dem Speicher entfernt.

Makrozugriff mit Access2Base

In LibreOffice ist seit der Version 4.2 die Erweiterung Access2Base integriert. Der Zugriff auf diese Bibliothek erfolgt über

```
    Sub DBOpen(Optional oEvent As Object)
        If GlobalScope.BasicLibraries.hasByName("Access2Base") then
            GlobalScope.BasicLibraries.loadLibrary("Access2Base")
        End If
        Call Application.OpenConnection(ThisDatabaseDocument)
    End Sub
```

Eine englischsprachige Beschreibung mit Beispielen ist auf der Seite *http://www.access2base.com/access2base.html* zu finden.

Die Bibliothek stellt nicht zusätzliche Funktionen zur Verfügung, sondern versucht, dem Anwender den Zugriff auf die Möglichkeiten der LibreOffice-API zu vereinfachen. Eine kurze Beschreibung ist auch in der Hilfe zu LO zu finden.