Base Handbuch

Kapitel 9 Makros

LibreOffice 24.2

Dokumentationen zu LibreOffice unter de.libreoffice.org

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 01.02.2024. Basierend auf der Version LibreOffice 24.2.

Inhalt

Allgemeines zu Makros 6
Der Makro-Editor 8
Benennung von Modulen, Dialogen und Bibliotheken 9
Makros in Base 10
Makros benutzen 10
Makros zuweisen 11
Ereignisse eines Formulars beim Öffnen oder Schließen des Fensters 11
Ereignisse eines Formulars bei geöffnetem Fenster 12
Ereignisse innerhalb eines Formulars 13
Bestandteile von Makros 14
Der «Rahmen» eines Makros 14
Variablen definieren 14
Arrays definieren 15
Zugriff auf das Formular 16
Zugriff auf Elemente eines Formulars 17
Zugriff auf die Datenbank 17
Die Verbindung zur Datenbank 17
SQL-Befehle 18
Vorbereitete SQL-Befehle mit Parametern 18
Datensätze lesen und benutzen 19
Mithilfe des Formulars 20
Ergebnis einer Abfrage 21
Mithilfe eines Kontrollfelds 22
Datensätze wechseln und bestimmte Datensätze ansteuern 23
Datensätze bearbeiten – neu anlegen, ändern, löschen 23
Inhalt eines Kontrollfelds ändern 23
Zeile einer Datenmenge ändern 24
Zeilen anlegen, ändern, löschen 24
Kontrollfelder prüfen und ändern 25
Englische Bezeichner in Makros 25
Eigenschaften bei Formularen und Kontrollfeldern 26
Schrift 26
Formular 26
Einheitlich für alle Arten von Kontrollfeld 27
Einheitlich für viele Arten von Kontrollfeld 27
Textfeld – weitere Angaben 27
Numerisches Feld 28
Datumsfeld 28
Zeitfeld 29
Währungsfeld 29
Formatiertes Feld 29
Listenfeld 30
Kombinationsfeld 31
Markierfeld, Optionsfeld 32
Maskiertes Feld 32
Tabellenkontrollfeld 32
Beschriftungsfeld 32
Gruppierungsrahmen 33
Schaltfläche 33
Navigationsleiste 33
Methoden bei Formularen und Kontrollfeldern 33
In einer Datenmenge navigieren 33

Datenzeilen bearbeiten ... 34
Einzelne Werte bearbeiten ... 36
Parameter für vorbereitete SQL-Befehle ... 37
Arbeit mit UNO-Befehlen in Formularen ... 37
Bedienbarkeit verbessern ... 38
Automatisches Aktualisieren von Formularen ... 38
Filtern von Datensätzen ... 39
Daten über den Formularfilter filtern ... 42
Filterdialog über einen Button starten ... 42
Durch Datensätze mit der Bildlaufleiste scrollen ... 44
Daten aus Textfeldern auf SQL-Tauglichkeit vorbereiten ... 45
Beliebige SQL-Kommandos speichern und bei Bedarf ausführen ... 46
Werte in einem Formular vorausberechnen ... 46
Die aktuelle Office-Version ermitteln ... 48
Wert von Listenfeldern ermitteln ... 48
Listenfelder durch Eingabe von Anfangsbuchstaben einschränken ... 49
Listenfelder mit eingeschränkter Auswahl ... 51
Listenfelder zur Mehrfachauswahl nutzen ... 54
Datumswert aus einem Formularwert in eine Datumsvariable umwandeln ... 56
Eingabemöglichkeiten in Feldern einschränken ... 56
Suchen von Datensätzen ... 57
Suchen in Formularen und Ergebnisse farbig hervorheben ... 59
Rechtschreibkontrolle während der Eingabe ... 63
Kombinationsfelder als Listenfelder mit Eingabemöglichkeit ... 65
Textanzeige im Kombinationsfeld ... 66
Fremdschlüsselwert vom Kombinationsfeld zum numerischen Feld übertragen ... 68
Kontrollfunktion für die Zeichenlänge der Kombinationsfelder ... 75
Datensatzaktion erzeugen ... 75
Navigation von einem Formular zum anderen ... 75
Datensatz im Formular direkt öffnen ... 77
Tabellen, Abfragen, Formulare und Berichte öffnen ... 78
Hierarchische Listenfelder ... 79
Filterung des Formulars mit hierarchischen Listenfeldern ... 80
Hierarchische Listenfelder in der Formulareingabe nutzen ... 83
Zeiteingaben mit Millisekunden ... 85
Ein Ereignis – mehrere Implementationen ... 86
Eingabekontrolle bei Formularen ... 87
Erforderliche Eingaben absichern ... 89
Fehlerhafte Eingaben vermeiden ... 92
Abspeichern nach erfolgter Kontrolle ... 94
Primärschlüssel aus Nummerierung und Jahreszahl ... 95
Datenbankaufgaben mit Makros erweitert ... 96
Verbindung mit Datenbanken erzeugen ... 96
Daten von einer Datenbank in eine andere kopieren ... 97
Direkter Import von Daten aus Calc ... 98
Zugriff auf Abfragen ... 103
Datenbanksicherungen erstellen ... 103
Interne Datenbanken sicher schließen ... 108

Tabellenindex heruntersetzen bei Autowert-Feldern 108
Drucken aus Base heraus 109
Druck von Berichten aus einem internen Formular heraus 109
Start, Formatierung, direkter Druck und Schließen des Berichts 109
Druck von Berichten aus einem externen Formular heraus 111
Serienbriefdruck aus Base heraus 112
Drucken über Textfelder 113
Drucken über Tabellen in Writer 114
Einfacher Druck von Text in Tabellen 114
Tabellendruck mit formatierten Zellen 116
Tabellendruck mit Bildern in der Tabelle 117
Rechnungen mit Übertrag im Tabellendruck 119
Aufruf von Anwendungen zum Öffnen von Dateien 120
Aufruf eines Mailprogramms mit Inhaltsvorgaben 122
Aufruf einer Kartenansicht zu einer Adresse 123
Mauszeiger ändern 124
Änderung beim Überfahren eines Links 124
Änderung bei gedrückter Strg-Taste und Mausklick 124
Formulare ohne Symbolleisten präsentieren 125
Formulare ohne Symbolleisten in einem Fenster 125
Formulare im Vollbildmodus 128
Formular direkt beim Öffnen der Datenbankdatei starten 128
Markierfelder durch Schaltflächen ersetzen 129
MySQL-Datenbank mit Makros ansprechen 130
MySQL-Code in Makros 130
Temporäre Tabelle als individueller Zwischenspeicher 131
Filterung über die Verbindungsnummer 131
Gespeicherte Prozeduren 131
Automatischer Aufruf einer Prozedur 132
Übertragung der Ausgabe einer Prozedur in eine temporäre Tabelle 132
PostgreSQL und Makros 133
Autowertrückgabe mit Returning 133
Datentyp «Array» 133
Dialoge 134
Dialoge starten und beenden 134
Einfacher Dialog zur Eingabe neuer Datensätze 136
Dialog zum Bearbeiten von Daten in einer Tabelle 138
Dialog zum Bearbeiten von Daten aus einer Tabellenübersicht 143
Fortschrittsbalken für den Ablauf mehrerer Prozeduren 149
Fehleinträge von Tabellen mit Hilfe eines Dialogs bereinigen 150
Makrozugriff mit Access2Base 158
Python als Makrosprache für Datenbanken 158
Speicherort für das erste Python Makro 158
Speicherort für Module zum Importieren in ein Makro 159
Verwaltung von Modulen in Bibliotheken 159
Abfrage an eine geöffnete Datenbankdatei 159
Abfrage an eine registrierte Datenbank 160
Fertige Module in die Base-Datei einbinden 161

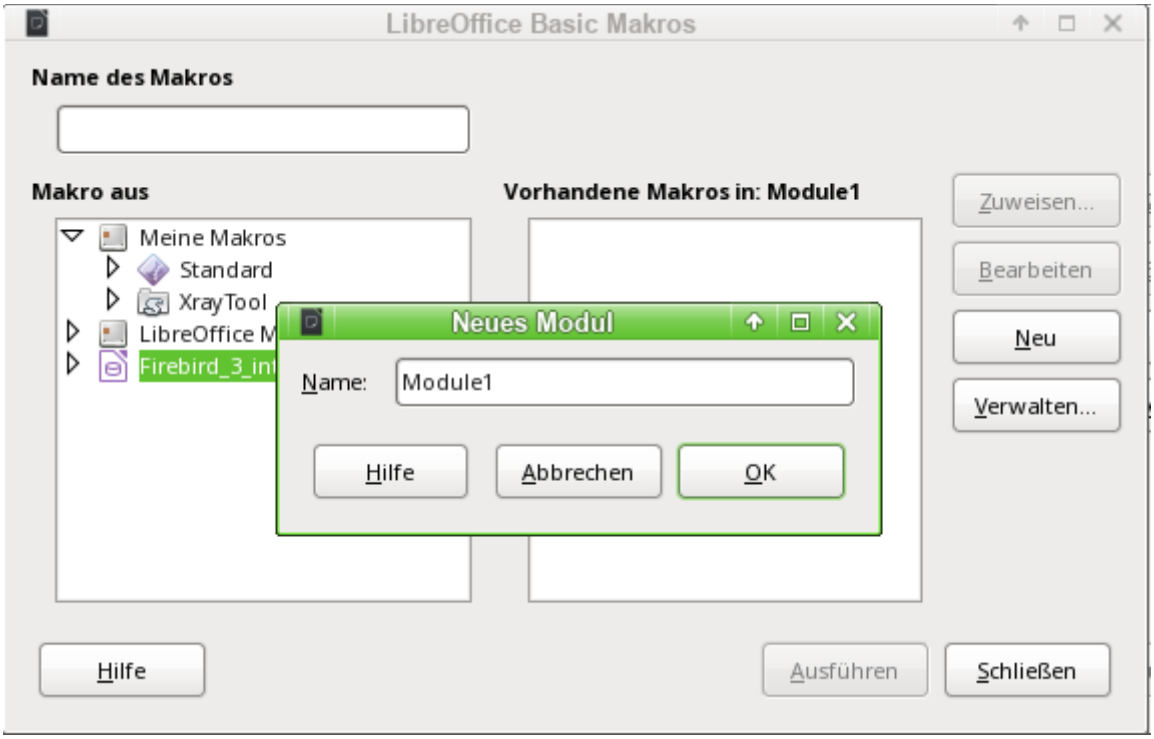
Allgemeines zu Makros

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:

```
001  REM  *****  BASIC  *****
002
003  Sub Main
004
005  End Sub
```

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

- Unter **Extras → Optionen → Sicherheit → Makrosicherheit** ist der **Sicherheitslevel → Mittel** zu wählen. Gegebenenfalls kann auch zusätzlich unter **Vertrauenswürdige Quellen → Vertrauenswürdige Speicherorte** 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 mit festem Variablentyp (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
```

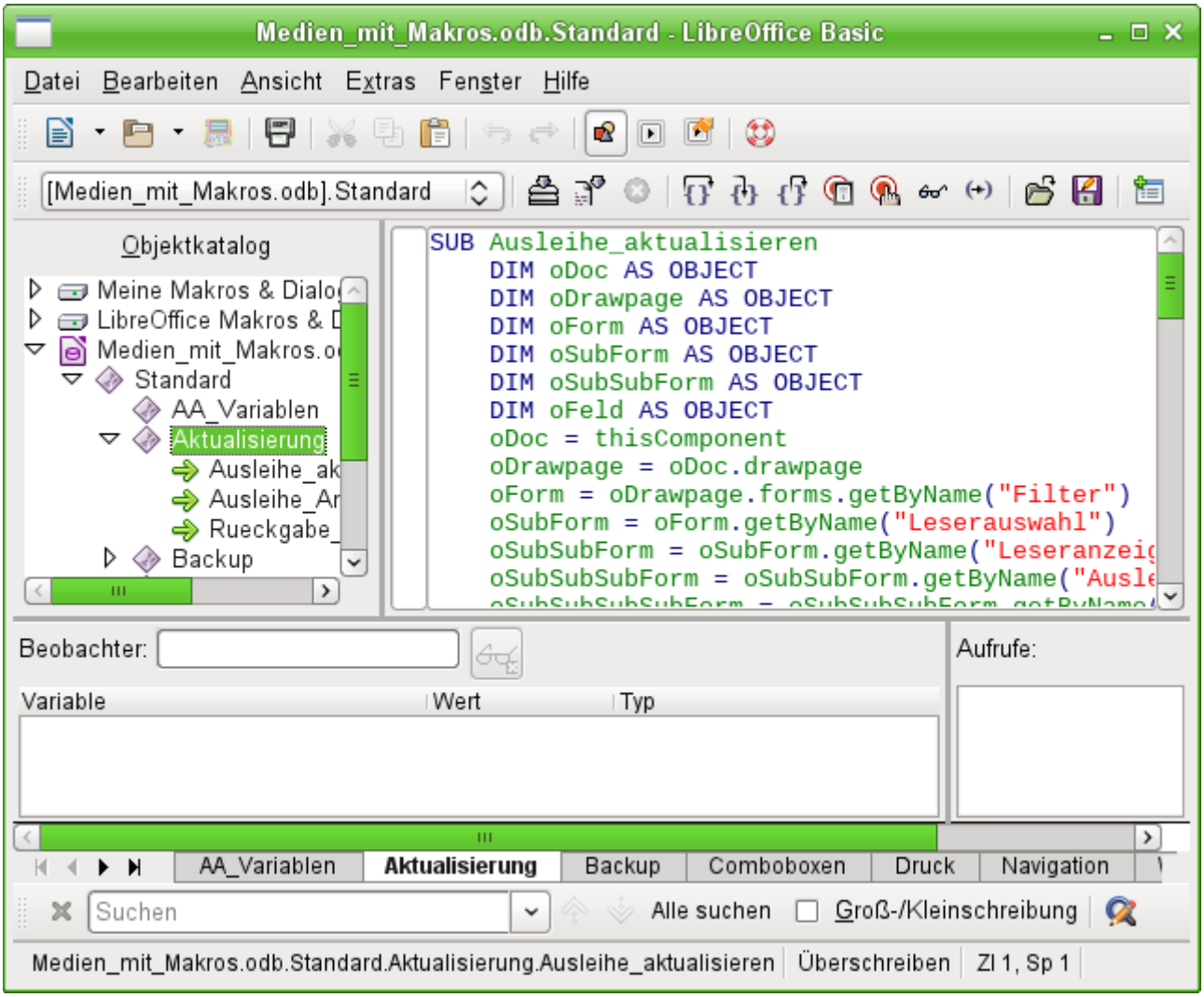
Hinweis

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.

Hinweis

Die hier aufgezeigten Makros sind **nahezu ausschließlich innerhalb der Base-Datei gespeichert** und dort auch getestet. So kann z.B. der Kontakt zu einer Datenbank mit **ThisDatabaseDocument** nur innerhalb einer Base-Datei hergestellt werden. Sollen die Makros außerhalb der Datei unter **Meine Makros und Dialoge** gespeichert werden, so kann eventuell statt **ThisDatabaseDocument** einfach **ThisComponent** zum Ziel führen. Es kann aber auch sein, dass dann bestimmte Methoden einfach nicht zur Verfügung stehen.

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 dort gemeinten Formular entsprechen.

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

Benennung von Modulen, Dialogen und Bibliotheken

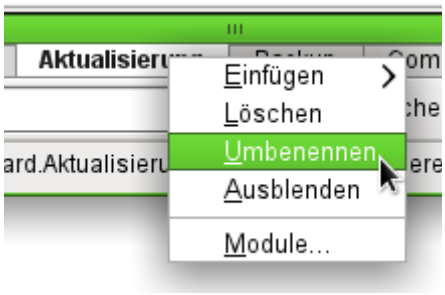
Die Benennung von Modulen, Dialogen und Bibliotheken sollte erfolgen, bevor irgendein Makro in die Datenbank eingebunden wird. Sie definieren schließlich den Pfad, in dem das auslösende Ereignis nach dem Makro sucht.

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

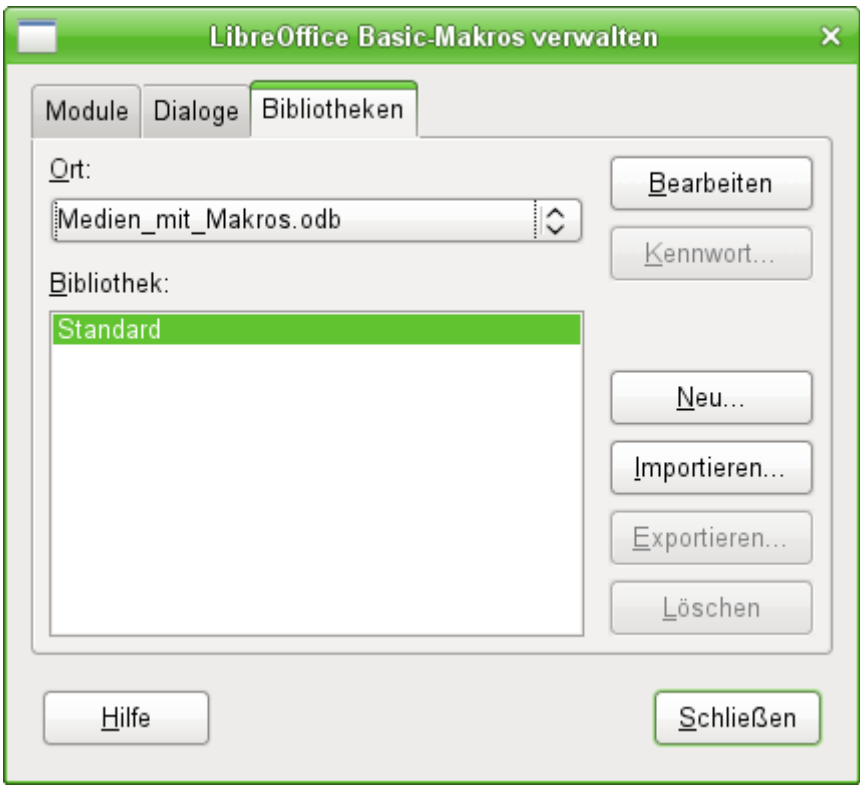
```
001 GlobalScope.BasicLibraries.LoadLibrary("Tools")
```

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

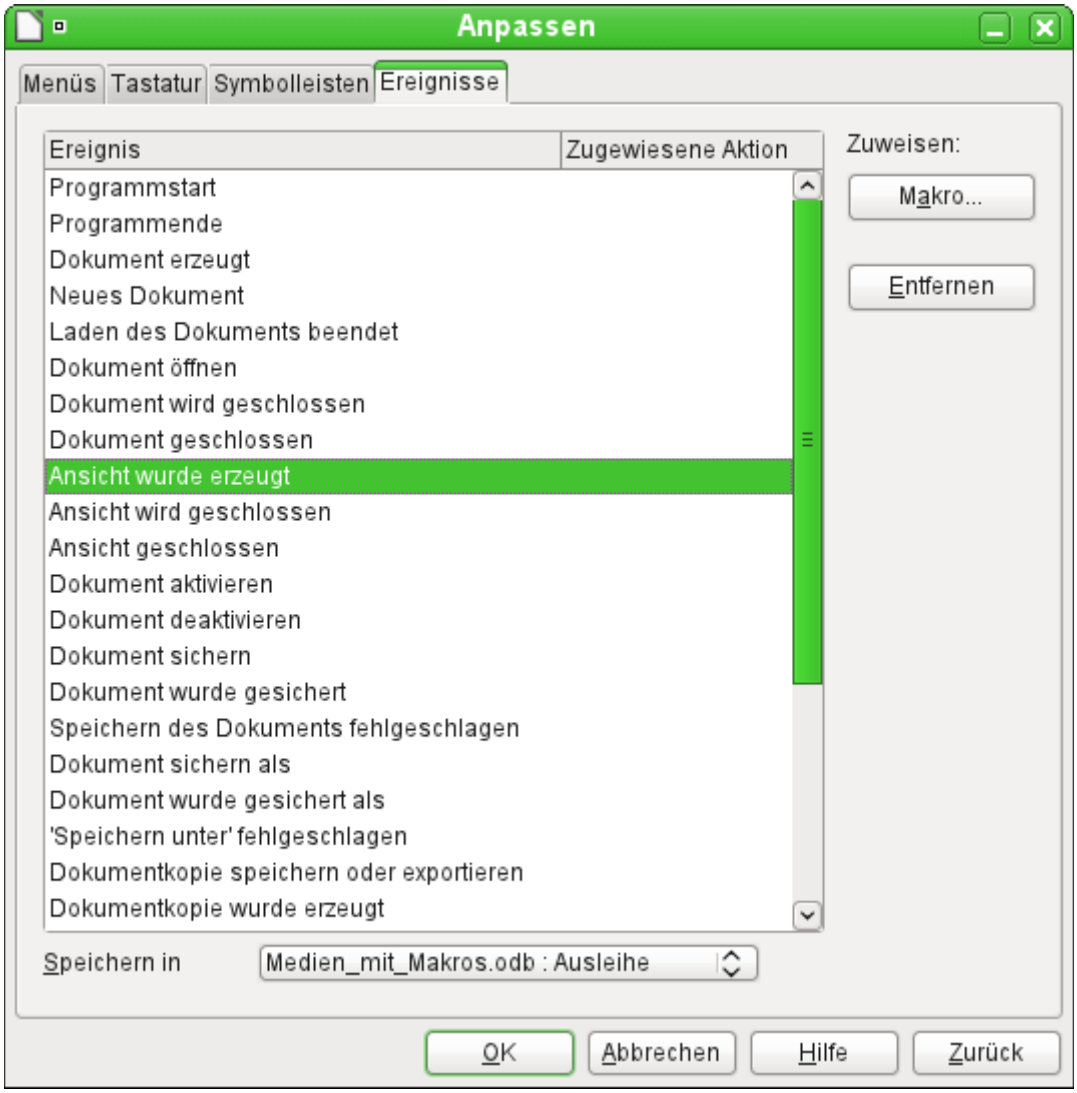
Der «direkte Weg» ist vor allem dann nicht möglich – auch nicht zu Testzwecken –, wenn eines der Objekte **`thisComponent`** (siehe den Abschnitt *Zugriff auf das Formular*) oder **`oEvent`** (siehe den Abschnitt *Zugriff auf Elemente eines Formulars*) benutzt wird.

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 Formulardokuments 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 (hier: «Medien_mit_Makros : Ausleihe)».

Dann kann diese Zuweisung mit OK bestätigt werden.

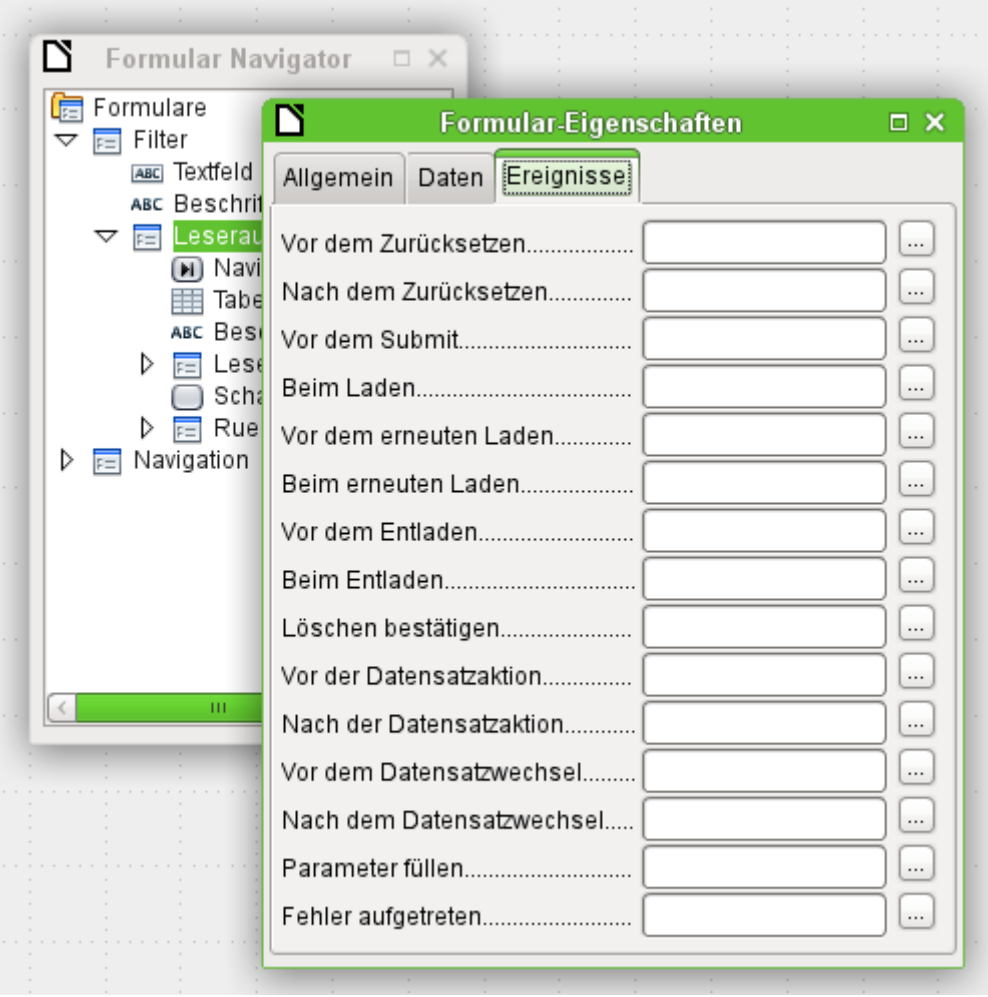
Die Einbindung von Makros in das Datenbankdokument erfolgt auf dem gleichen Weg. Nur stehen hier teilweise andere Ereignisse zur Wahl.

Vorsicht

Beim Starten eines Datenbankdokumentes kann, wenn die Makrounterstützung nicht aktiviert wurde, die Nachfrage kommen, ob Makros ausgeführt werden. Wird in einem Datenbankdokument das Ereignis **Ansicht wurde erzeugt** gewählt, so wird dieses Ereignis nicht ausgeführt, auch wenn die Frage nach dem Ausführen der Makros bejaht wurde.
Das Ereignis **Dokument öffnen** wird hingegen ausgeführt. Soll also auf jeden Fall beim Start des Datenbankdokumentes ein Makro ausgeführt werden, so sollte dieses Ereignis gewählt werden.

Ereignisse eines Formulars bei geöffnetem Fenster

Nachdem das Fenster für die gesamten Inhalte des Formulars (Formulardokument) geöffnet wurde, kann auf die einzelnen Elemente des Formulardokuments 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.

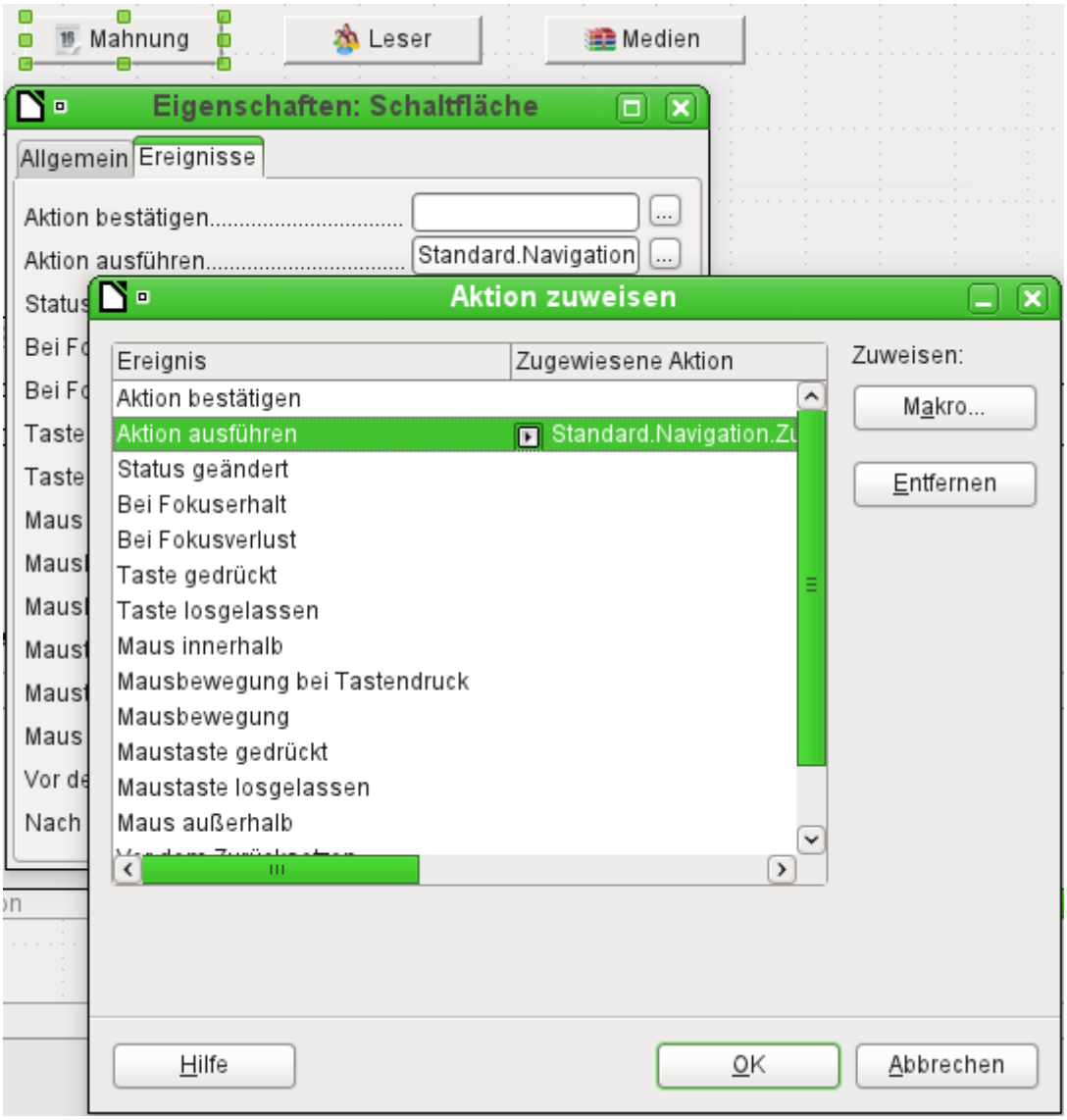
Die unter **Formular-Eigenschaften → Ereignisse** aufgezeigten Ereignisse finden statt während das Formularfenster geöffnet ist. Sie können für jedes Formular oder Unterformular des Formularfensters separat ausgewählt werden.

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.
Das Formularfenster wird in dieser Dokumentation zur besseren Unterscheidung deshalb oft auch als «Formulardokument» bezeichnet.

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.

```
001  SUB Ausleihe_aktualisieren
002  END SUB
```

```
001  SUB Zu_Formular_von_Formular(oEvent AS OBJECT)
002  END SUB
```

```
001  FUNCTION Loeschen_bestaetigen(oEvent AS OBJECT) AS BOOLEAN
002     Loeschen_bestaetigen = FALSE
003  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.

```
001  DIM oDoc AS OBJECT
002  DIM oDrawpage AS OBJECT
003  DIM oForm AS OBJECT
004  DIM sName AS STRING
005  DIM bOKEnabled AS BOOLEAN, iCounter AS INTEGER, dBirthday AS DATE
```

Die Deklaration von Variablen ist in Einzelzeilen oder zusammengefasst in einer Zeile, getrennt durch ein Komma, möglich.

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.

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.

```
001 DIM arDaten()
```

erzeugt eine leeres Array.

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

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

Über

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

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

```
001 DIM arDaten(2)
002 arDaten(0) = "Lisa"
003 arDaten(1) = "Schmidt"
004 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.

```
005 ReDIM Preserve arDaten(3)
006 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.

```
001 DIM arDaten(2,1)
002 arDaten(0,0) = "Lisa"
003 arDaten(1,0) = "Schmidt"
004 arDaten(2,0) = "Köln"
005 arDaten(0,1) = "Egon"
006 arDaten(1,1) = "Müller"
007 arDaten(2,1) = "Hamburg"
```

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.

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:

```
001  oDoc = thisComponent
002  oDrawpage = oDoc.drawpage
003  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:

```
004  DIM oSubForm AS OBJECT
005  DIM oSubSubForm AS OBJECT
006  oSubForm = oForm.getByName("Leserauswahl")
007  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.)

```
006  oForm = thisComponent.drawpage.forms.getByName("Filter")
007  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:

```
006  oForm = thisComponent.drawpage.forms.Filter
007  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:

```
001  SUB MakrobeispielBerechne(oEvent AS OBJECT)
002     oForm = oEvent.Source
003     ...
004  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:

```
001  SUB MakrobeispielBerechne(oEvent AS OBJECT)
002     oFeld = oEvent.Source.Model
003     oForm = oFeld.Parent
004     ...
005  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:

```
001  DIM btnOK AS OBJECT  ' Button »OK»
002  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:

```
001  SUB Auswahl_bestaetigen(oEvent AS OBJECT)
002     DIM btnOK AS OBJECT
003     btnOK = oEvent.Source.Model
004  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:

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

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

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

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

```
001  DIM oDatasource AS OBJECT
002  DIM oConnection AS OBJECT
003  oDatasource = thisComponent.Parent.CurrentController
004  IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
005  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. Dazu muss das Makro innerhalb der Base-Datei gespeichert werden.

```
001  DIM oDatasource AS OBJECT
002  DIM oConnection AS OBJECT
003  oDatasource = thisDatabaseDocument.CurrentController
004  IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
005  oConnection = oDatasource.ActiveConnection()
```

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

```
001  DIM oDatabaseContext AS OBJECT
002  DIM oDatasource AS OBJECT
003  DIM oConnection AS OBJECT
004  oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
005  oDatasource = oDatabaseContext.getByName("angemeldeter Name der Datenbank in LO")
006  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:

```
001  DIM oSQL_Statement AS OBJECT    ' das Objekt, das den SQL-Befehl ausführt
002  DIM stSql AS STRING             ' Text des eigentlichen SQL-Befehls
003  DIM oResult AS OBJECT           ' Ergebnis für executeQuery
004  DIM iResult AS INTEGER          ' Ergebnis für executeUpdate
005  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.

```
006  stSql = "SELECT * FROM ""Tabelle1"""
007  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.

```
008  stSql = "DROP TABLE ""Suchtmp"" IF EXISTS"
009  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.

Hinweis

SQL-Befehle, die so abgesandt werden, entsprechen nicht genau dem, was z. B. bei den Abfragen über direkte SQL-Ausführung erreicht wird. Eine Abfrage wie
… "Name" LIKE '%*%'
gibt nicht nur die Namen mit einem '*' wieder, da intern aus dem '*' ein '%' erstellt wird. Um wirklich das gleiche Verhalten mit direkter SQL-Ausführung zu erhalten, muss
oSQL_Statement.EscapeProcessing = False
nach der Erstellung von oSQL_Statement und vor der Ausführung des Codes eingefügt werden:

```
001  oSQL_Statement = oConnection.createStatement()
002  oSQL_Statement.EscapeProcessing = False
```

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:

```
001  DIM oSQL_Statement AS OBJECT      ' das Objekt, das den SQL-Befehl ausführt
002  DIM stSql AS STRING               ' Text des eigentlichen SQL-Befehls
003  stSql = "UPDATE ""Verfasser"" " _
004     & "SET ""Nachname"" = ?, ""Vorname"" = ?" _
005     & "WHERE ""ID"" = ?"
006  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:

```
007  oSQL_Statement.setString(1, oTextfeld1.Text)    ' Text für den Nachnamen
008  oSQL_Statement.setString(2, oTextfeld2.Text)    ' Text für den Vornamen
009  oSQL_Statement.setLong(3, oZahlenfeld1.Value)   ' Wert für die betreffende ID
010  iResult = oSQL_Statement.executeUpdate
```

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

Es ist auch möglich, direkt mehrere Datensätze über einen vorbereiteten SQL-Befehl einzufügen:

```
011  stSql = "UPDATE ""Person"" " _
012     & "SET ""Name"" = ?" _
013     & "WHERE ""ID"" = ?"
014  oSQL_Statement = oConnection.prepareStatement(stSql)
015  oSQL_Statement.setString(1, "Bill")
016  oSQL_Statement.setLong(2, 1)
017  oSQL_Statement.addBatch()
018  oSQL_Statement.setString(1, "Michaela")
019  oSQL_Statement.setLong(2, 2)
020  oSQL_Statement.addBatch()
021  oSQL_Statement.executeBatch()
```

Mit **clearBatch** könnte der gesamte Inhalt vor der Ausführung auch wieder zurückgenommen werden.

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 um 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, Ansicht (*View*)) anzeigt. Dafür gibt es mehrere Methoden, die mit **get** und dem Datentyp bezeichnet sind, beispielsweise diese:

```
001 DIM ID AS LONG
002 DIM sName AS STRING
003 DIM dValue AS CURRENCY
004 DIM dEintritt AS NEW com.sun.star.util.Date
005 ID = oForm.getLong(1)
006 sName = oForm.getString(2)
007 dValue = oForm.getDouble(4)
008 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.

Soll anstelle der Spaltennummern mit den Spaltennamen der zugrundeliegenden Datenmenge (Tabelle, Abfrage, Ansicht (*View*)) gearbeitet werden, so kann die Spaltennummer über die Methode **findColumn** ermittelt werden. Hier ein Beispiel zum Auffinden der Spalte "Name":

```
001 DIM sName AS STRING
002 DIM nName AS STRING
003 nName = oForm.findColumn("Name")
004 sName = oForm.getString(nName)
```

Das Ergebnis ist immer ein 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.

```
001  SUB WerteAuslesen(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM stFeld1 AS STRING
004     oForm = oEvent.Source.Model.Parent
005     stFeld1 = oForm.getString(1)
006  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.

Hinweis

Grundsätzlich können alle Felder mit **getString** ausgelesen werden. Um sie in Makros weiter zu verarbeiten, müssen die erhaltenen Strings dann gegebenenfalls anschließend in die jeweiligen Variablen umgewandelt werden.
Das Auslesen mit **getString** hat den Vorteil, dass auch leere Felder einwandfrei ermittelt werden können. Werden die (passenden) Variablen stattdessen z.B. über **getInt** ausgelesen, so ergibt selbst ein leeres Feld '0'. Dies täuscht also vor, dass in dem Feld eben diese Zahl enthalten war. So etwas ist besonders lästig, wenn eben auch der Wert '0' selbst vorkommt – bei der internen HSQLDB z. B. beim automatisch hoch zählenden Schlüsselwert als Startwert.
Alternativ muss immer mit **wasNull** nachgesehen werden, ob denn der Inhalt vielleicht leer gewesen ist.

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:

```
001  WHILE oResult.next         ' einen Datensatz nach dem anderen verarbeiten
002     rem übernimm die benötigten Werte in Variablen
003     stVar = oResult.getString(1)
004     inVar = oResult.getLong(2)
005     boVar = oResult.getBoolean(3)
006     rem mach etwas mit diesen Werten
007  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

```
001  oResult.next
```

zuerst die Zeile auf diesen Datensatz bewegt und dann mit

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

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

```
003  IF wasNull THEN
```

```
004     ...
005  END IF
```

Mit dieser Nachfrage würde die gerade getätigte Abfrage zu **stVar** darauf überprüft, ob sie nach SQL-Standard **NULL** gewesen ist.

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

```
001  oSQL_Anweisung.ResultSetType = 1004
```

oder

```
001  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

```
001  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:

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

Hinweis

Sollen viele Daten über die Schleife **WHILE oResult.next** ausgelesen werden, so macht sich hier ein **Geschwindigkeitsunterschied** bei verschiedenen Datenbanken deutlich bemerkbar. Die interne Firebird Datenbank arbeitet hier deutlich schneller als die interne Hsqldb. Selbst bei gleichen Datenbanken kann es abhängig vom Treiber zu deutlichen Unterschieden kommen. Bei der MariaDB mit direkter Verbindung ist die Geschwindigkeit auf dem Level der Firebird Datenbank. Mit der JDBC-Verbindung hingegen ist das Auslesen so langsam wie mit der Hsqldb.
Dieses Verhalten kann schon bei einer Abfrage getestet werden, wenn zum Ausführen die direkte SQL-Verbindung genutzt wird. Das Scrollen zum letzten Datensatz braucht bei vielen Zeilen deutlich länger mit der internen Hsqldb und MariaDB über die JDBC-Verbindung als mit der internen Firebird Datenbank und MariaDB mit der direkten Verbindung.

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:

```
001  sValue = oTextField.BoundField.Text        ' Beispiel für ein Textfeld
```

```
002  nValue = oNumericField.BoundField.Value    ' Beispiel für ein numerisches Feld
003  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:

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

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

Die Methode mit **getRow()** funktioniert aber nur dann einwandfrei, wenn es sich bei dem aktuellen Datensatz nicht um einen neuen Datensatz handelt. Deswegen hier eine weitere Möglichkeit der Navigation über den **Bookmark**-Befehl. Damit lässt sich der Cursor auch dann sicher positionieren, wenn bei einer neuen Zeile **getRow() 0** ermittelt.

```
001  DIM var AS VARIANT
002  var = oForm.getBookmark()
003  loRow = oForm.getRow()
004  oForm.reload()
005  IF loRow = 0 THEN
006     oForm.MoveToInsertRow()
007  ELSE
008     oForm.MoveToBookmark(var)
009  END IF
```

Bei einem neuen Datensatz wird die Zeile mit '0' angegeben. In diesem Fall wird dann der Cursor nach dem Neuladen mit **MoveToInsertRow** auf diese Position gesetzt. Mit **MoveToBookmark** würde hier der Cursor auf die letzte Zeile mit Inhalt gesetzt.

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. Diese Eigenschaft dient nur zur Anzeige des Wertes.

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:

```
001  DIM unoDate AS NEW com.sun.star.util.Date
```

```
002  unoDate.Year = Year(Date)
003  unoDate.Month = Month(Date)
004  unoDate.Day = Day(Date)
005  oDateField = oForm.getByName("Datum")
006  oDateField.BoundField.updateDate( unoDate )
007  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:

```
001  oDateField = oForm.getByName("Datum")
002  oDateField.BoundField.updateDate( CDateToUnoDate(NOW()) )
003  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:

```
001  DIM unoDate AS NEW com.sun.star.util.Date
002  unoDate.Year = Year(Date)
003  unoDate.Month = Month(Date)
004  unoDate.Day = Day(Date)
005  oForm.updateDate(3, unoDate )
006  oForm.updateString(4, "ein Text")
007  oForm.updateDouble(6, 3.14)
008  oForm.updateInt(7, 16)
009  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**).

```
001  IF oForm.isNew THEN
002     oForm.insertRow()
003  ELSE
004     oForm.updateRow()
005  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:

```
001  SUB Main(oEvent AS OBJECT)
002     DIM stTag AS STRING
003     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:

```
001  SUB showWarning(oField AS OBJECT, iType AS INTEGER)
002     SELECT CASE iType
003        CASE 1
004           oField.TextColor = RGB(0,0,255)    ' 1 = blau
005        CASE 2
006           oField.TextColor = RGB(255,0,0)    ' 2 = rot
007        CASE ELSE
008           oField.TextColor = RGB(0,255,0)    ' 0 = grün (weder 1 noch 2)
009     END SELECT
010  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* .

```
001  SUB Main(oEvent AS OBJECT)
002     Xray(oEvent)
003  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».

Wenn mit

```
001  oFeld = oForm.getByName("Name des Kontrollfeldes")
```

auf ein Kontrollfeld zugegriffen wird, so werden die Eigenschaften einfach durch ein Anhängen an dieses Objekt mit einem Punkt als Verbinder angesprochen:

```
001  oFeld.FontHeight = 16
```

definiert also z. B. die Schriftgröße in 16 Punkten.

Schrift

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

Name	***Daten-typ***	***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	Daten-typ	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	Daten-typ	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	Daten-typ	L/S	Eigenschaft
String	string	L+S	Inhalt des Feldes aus der Anzeige.
MaxTextLen	integer	L+S	Maximale Textlänge.

Name	Daten-typ	L/S	Eigenschaft
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	Daten-typ	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	Daten-typ	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 Formulardefinition oder der *API-Referenz* zu entnehmen.

Name	Daten-typ	Datentyp ab LO 4.1.1	L/S	Eigenschaft
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	Daten-typ	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	Daten-typ	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:

```
001  oControl = oForm.getByName("Name des Listenfelds")
002  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.

```
001  oControl = oForm.getByName("Name des Listenfelds")
002  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:

```
001  SUB Listenfeldfilter
002     DIM stSql(0) AS STRING
003     DIM oDoc AS OBJECT
004     DIM oDrawpage AS OBJECT
005     DIM oForm AS OBJECT
006     DIM oFeld AS OBJECT
007     oDoc = thisComponent
008     oDrawpage = oDoc.drawpage
009     oForm = oDrawpage.forms.getByName("MainForm")
010     oFeld = oForm.getByname("Listenfeld")
011     stSql(0) = "SELECT ""Name"", ""ID"" FROM ""Filter_Name"" ORDER BY ""Name"""
012     oFeld.ListSource = stSql
013     oFeld.refresh
014  END SUB
```

Hinweis

Soll der gerade geänderte Wert eines Listenfeldes ausgelesen werden, der noch nicht im Formular abgespeichert ist, so geht dies über die Listenposition:

```
001  SUB Kontofilter_Feldstart(oEvent AS OBJECT)
002     DIM oFeld AS OBJECT
003     DIM inID AS INTEGER
004     oFeld = oEvent.Source.Model
005     inID = oFeld.ValueItemList(oEvent.Selected)
006     ...
007  END SUB
```

Statt **`oEvent.Selected`** kann hier natürlich auch **`oFeld.SelectedItemPos`** stehen.

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.

Name	Datentyp	L/S	Eigenschaft
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.

Name	Datentyp	L/S	Eigenschaft
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).
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	Daten-typ	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.

Name	Datentyp	Beschreibung
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.
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 Werte der aktuellen Zeile neu ein. Nach dem Abspeichern der Zeile sind das auch die aktuellen Werte.
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.

Name	***Datentyp***	***Beschreibung***
Werte lesen		
getString(c)	string	Liefert den Inhalt der Spalte als Zeichenkette. Kann auch zum Auslesen aller anderen Spalten genutzt werden und hat den Vorteil, dass leere Felder direkt bestimmt werden können. Strings können immer noch später in andere Datentypen umgewandelt werden.
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.
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.		
getBinaryStream(c)	Object	Liest den Inhalt eines Binärfeldes (z.B. Bild) aus. So ein Inhalt könnte als Datei gespeichert werden.
wasNull	boolean	Gibt an, ob der Wert der zuletzt gelesenen Spalte NULL war. Beim Auslesen z.B. mit getInt wird sonst für ein leeres Feld grundsätzlich '0' weitergegeben.
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)	–	

Name	Datentyp	Beschreibung
updateString(c,s)	–	Speichert in Spalte c die angegebene Zeichenkette s.
updateBytes(c,x)	–	Speichert in Spalte c das angegebene Byte-Array x.
updateDate(c,d)	–	Speichert in Spalte c das angegebene Datum d.
updateTime(c,d)	–	Speichert in Spalte c den angegebenen Zeitwert d.
updateTimestamp(c,d)	–	Speichert in Spalte c den angegeb. 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.		
getBinaryStream(c)	Object	Liest den Inhalt eines Binärfeldes (z.B. Bild) aus. So ein Inhalt könnte als Datei gespeichert werden.
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.

Name	Datentyp	Beschreibung
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.

Arbeit mit UNO-Befehlen in Formularen

Über den Makrorekorder können die Befehle ausgelesen werden, die z. B. mit den Buttons aus der Navigationsleiste der Formulare verbunden sind. Diese Befehle haben häufig eine umfassendere Funktion als die Funktionen, die sonst für Makros vorgesehen sind.

```
001  SUB FormularNeuLadenKontrollfelderAktualisieren
002     DIM oDocument AS OBJECT
```

```
003     DIM oDispatcher AS OBJECT
004     DIM Array()
005     oDocument = ThisComponent.CurrentController.Frame
006     oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
007     oDispatcher.executeDispatch(oDocument, ".uno:Refresh", "", 0, Array())
008 END SUB
```

Über den **Dispatcher** wird das Formular neu geladen und die Formularfelder aktualisiert. Würde nur das Formular über **oForm.reload()** neu eingelesen, nachdem aus einem anderen Formular heraus der Inhalt eines Listenfeldes geändert wurde, so würde die Änderung in dem Formular nicht angezeigt. Hier müsste auch noch jedes einzelne Feld mit **oFeld.refresh()** neu eingelesen werden.

Auch das Sichern eines Datensatz ist über **.uno:RecSave** einfacher als mittels der direkten Ansprache des Formulars. Bei der direkten Ansprache des Formulars muss erst geklärt werden, ob es sich um einen neuen Datensatz handelt, für den dann ein **Insert** durchgeführt wird, oder ob es sich um einen bestehenden Datensatz handelt, der dann ein **Update** erfordert.

Eine Übersicht über verschiedene UNO-Befehle befindet sich im Anhang des Handbuches. UNO-Befehle können auch z. B. über **Extras → Anpassen → Symbolleisten → Beschreibung** ermittelt werden.

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.

```
001 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 durchgeführt, die nach außen in der Regel keinen Wert zurück gibt. 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 standardmäßig 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.

```
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     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".

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

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

```
008     oForm.reload()
009 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 Abspeicherbutton und liest die Listenfelder neu ein, so dass ein gewählter Filter aus einem Listenfeld die Auswahl in dem anderen Listenfeld einschränken kann.[2]

```
001 SUB Filter
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm1 AS OBJECT
005     DIM oForm2 AS OBJECT
006     DIM oFeldList1 AS OBJECT
007     DIM oFeldList2 AS OBJECT
008     oDoc = thisComponent
009     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.

```
010     oForm1 = oDrawpage.forms.getByName("Filter")
011     oForm2 = oDrawpage.forms.getByName("Anzeige")
012     oFeldList1 = oForm1.getByName("Liste_1")
013     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-Kommando überschrieben.

```
014     oFeldList1.commit()
015     oFeldList2.commit()
016     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.

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

```
017     oFeldList1.refresh()
018     oFeldList2.refresh()
019     oForm2.reload()
020 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.

```
001 SELECT
002    "Feld_1" || ' - ' || "Anzahl" AS "Anzeige",
003    "Feld_1"
004 FROM
005    ( SELECT COUNT( "ID" ) AS "Anzahl", "Feld_1" FROM "Suchtabelle"
          GROUP BY "Feld_1" )
006 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.

```
001 SELECT
002    COALESCE( "Feld_1" || ' - ' || "Anzahl", 'leer - ' || "Anzahl" )
          AS "Anzeige",
003    "Feld_1"
004 FROM
005    ( SELECT COUNT( "ID" ) AS "Anzahl", "Feld_1" FROM "Tabelle"
006    WHERE "ID" IN
007       ( SELECT "Tabelle"."ID" FROM "Filter", "Tabelle"
008          WHERE "Tabelle"."Feld_2" = COALESCE( "Filter"."Filter_2",
009          "Tabelle"."Feld_2" ) )
010    GROUP BY "Feld_1"
011    )
012 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 Lis-

tenfelder 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:

```
001 SELECT "Tabelle"."ID"
002 FROM "Filter", "Tabelle"
003 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:

```
001 SELECT "Tabelle"."ID"
002 FROM "Filter", "Tabelle"
003 WHERE "Tabelle"."Feld_2" = COALESCE( "Filter"."Filter_2",
          "Tabelle"."Feld_2" )
004    AND
005    "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.

```
001 SUB Filter_Zusatzinfo(oEvent AS OBJECT)
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm1 AS OBJECT
005    DIM oForm2 AS OBJECT
006    DIM stTag AS String
007    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.

```
008    aList() = Split(stTag, ",")
009    oDoc = thisComponent
010    oDrawpage = oDoc.drawpage
011    oForm1 = oDrawpage.forms.getByName("Filter")
012    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.

```
013    FOR i = LBound(aList()) TO Ubound(aList())
014       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.

```
015             oForm1.getByName(aList(i)).commit()
016             oForm1.updateRow()
017         ELSE
```

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

```
018             oForm1.getByName(aList(i)).refresh()
019         END IF
020     NEXT
021     oForm2.reload()
022 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.

```
001 SUB FilterSetzen
002     DIM oDoc AS OBJECT
003     DIM oForm AS OBJECT
004     DIM oFeld  AS OBJECT
005     DIM stFilter As String
006     oForm = thisComponent.Drawpage.Forms.getByName("MainForm")
007     oFeld = oForm.getByName("Filter")
008     stFilter = oFeld.Text
009     oForm.filter = " UPPER(""Name"") LIKE '%'||'" + UCase(stFilter) + "'||'%'"
010     oForm.ApplyFilter = TRUE
011     oForm.reload()
012 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.

```
001 SUB FilterEntfernen
002     DIM oForm AS OBJECT
003     oForm = thisComponent.Drawpage.Forms.getByName("MainForm")
004     oForm.ApplyFilter = False
005     oForm.reload()
006 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.

Filterdialog über einen Button starten

Wird ein Formular geöffnet, so stehen über die Navigationsleiste verschiedene Filtermöglichkeiten zur Verfügung. Während in Tabellen, Abfaregn und über das Formularkontrollelement ein Filterdialog zur Verfügung steht, ist dieser über die Navigationsleiste des Formularfensters durch den Formularbasierten Filter ersetzt worden. Auch über eine einfache Schaltfläche lässt sich dieser Dialog nicht starten. Hier kann zur Zeit nur mit einem Makro nachgeholfen werden.

```
001 SUB FilterDefault(oEvent AS OBJECT)
002     oForm = oEvent.Source.Model.Parent
```

```
003     oController = thisComponent.getCurrentController()
004     oFormController = oController.getFormController(oForm)
005     oFormController.FormOperations.execute(16)
006 END SUB
```

Das Makro wird von der Schaltfläche aus über **Ereignisse → Aktion ausführen** gestartet. Über das auslösende Ereignis wird das zugrundeliegende Formular ermittelt. Anschließend wird über den Controller die Kontrolle über die Elemente des Formulars übernommen, die dort prinzipiell zur Verfügung stehen. Zu den möglichen FormOperations gehört mit der Short-Variablen 16 der Dialog zum Setzen eines Filters.

Der Filterdialog kann über den Button gestartet werden und die Filterundg wird mit OK direkt vollzogen.

Die folgenden Variablen stehen über die com::sun::star::form::runtime::FormFeature Constant Group Reference zur Verfügung:

Nr.	***Befehl***	***Bedeutung***

1	MoveAbsolute
	`005 DIM v as new com.sun.star.beans.NamedValue`
	`006 v.Name = "Position"`
	`007 v.Value = 5`
	`008 oFormController.FormOperations.executeWith Arguments(1, Array(v))`
		Gehe zu dem Datensatz, der über ein Array angegeben werden muss

2	TotalRecord
	`005 oState = oFormController.FormOperations.getState(2)`
	`006 Msgbox "Gesamtzahl der Datensätze: "& cStr(oState.State)`
		Zeige die Anzahl der gesamten Datensätze an – leider auch hier nur wie in der Navigationsleiste mit '41 *'

3	MoveToFirst	Gehe zum ersten Datensatz
4	MoveToPrevious	Gehe zum vorhergehenden Datensatz
5	MoveToNext	Gehe zum nächsten Datensatz
6	MoveToLast	Gehe zum letzten Datensatz
7	MoveToInsertRow	Gehe zum Einfügen eines neuen Datensatzes
8	SaveRecordChanges	Speichere die Datensatzänderung
9	UndoRecordChanges	Mache die Änderungen rückgän-

		gig
10	DeleteRecord	Lösche den Datensatz
11	ReloadForm	Lies das Formular neu ein
12	SortAscending	Sortiere aufsteigend
13	SortDescending	Sortiere absteigend
14	InteractiveSort	Öffne Dialog zum Sortieren
15	AutoFilter	Automatischer Filter, vorher in ein gewünschtes Feld klicken.
16	InteractiveFilter	Öffne Dialog zum Filtern
17	ToggleApplyFilter	Stelle den Filter ein oder aus
18	RemoveFilterAndSort	Nimm Filterung und Sortierung zurück
19	RefreshCurrentControl	Lies den Inhalt des Listenfeldes oder Kombinationsfeldes neu ein.

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.

```
001 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.

```
001 SUB MaxRow(oEvent AS OBJECT)
002    'Auslösen durch das Formular "Beim Laden" und "Nach der Datensatzaktion"
003    DIM oForm AS OBJECT
004    DIM oScrollField AS OBJECT
005    DIM loMax AS LONG
006    oForm = oEvent.Source
007    oScrollField = oForm.getByName("Bildlaufleiste")
008    loPos = oForm.getRow
009    oForm.last
010    loMax = oForm.getRow
011    oForm.absolute(loPos)
012    oScrollField.ScrollValueMax = loMax
013    oScrollField.ScrollValue = loPos
014 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.

```
001 SUB Navigation(oEvent AS OBJECT)
002    'Auslösen durch das Formular "Nach dem Datensatzwechsel"
003    'Synchronisiert die Scrollstellung mit der Position des Datensatzes
```

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

```
004     DIM oForm AS OBJECT
005     DIM oScrollField AS OBJECT
006     oForm = oEvent.Source
007     oScrollField = oForm.getByName("Bildlaufleiste")
008     loPos = oForm.getRow
009     IF loPos = 0 THEN
010        'Bei einem neuen Datensatz wird über getRow '0' ermittelt.
011        'In dem Datensatzanzeiger soll stattdessen die maximale Zahl an Zeilen
012        ''RowCount' um '1' erhöht werden
013        loPos = oForm.RowCount + 1
014     END IF
015     oScrollField.ScrollValue = loPos
016  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.

```
001  SUB FormScroll(oEvent AS OBJECT)
002     'Auslösen durch die Bildlaufleiste "Beim Justieren"
003     DIM oForm AS OBJECT
004     DIM oScrollAction AS OBJECT
005     oScrollAction = oEvent.Source
006     oForm = oScrollAction.Model.Parent
007     loPos = oScrollAction.getValue()
008     oForm.absolute(loPos)
009  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.

```
001  FUNCTION String_to_SQL(st AS STRING)
002     IF InStr(st,"'") THEN
003        st = Replace(st,"'","''")
004     END IF
005     String_to_SQL = st
006  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 so wird das eine Hochkomma durch zwei Hochkommata ersetzt. Der SQL-Code wird so maskiert.

Die Funktion übergibt ihr Ergebnis durch den folgenden Aufruf:

```
001  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:

```
001  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.

```
001 SUB ChangeData(oEvent AS OBJECT)
002    DIM oConnection AS OBJECT
003    DIM oForm AS OBJECT
004    DIM stSql AS STRING
005    DIM oSql_Statement AS OBJECT
006    DIM inValue AS INTEGER
007    oForm = oEvent.Source.Model.Parent
008    oConnection = oForm.activeConnection()
009    stSQL = oForm.getString(2)
010    inValue = MsgBox("Soll der SQL-Code" & CHR(13) & stSQL & CHR(13) &
          "ausgeführt werden?", 20, "SQL-Code ausführen")
011    IF inValue = 6 THEN
012       oSQL_Statement = oConnection.createStatement()
013       oSQL_Statement.execute(stSql)
014    END IF
015 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.

Hinweis

Zur Ausführung eines SQL-Kommados stehen die folgenden Methoden zur Verfügung[5]:
executeQuery(stSql) erwartet eine Rückgabe eines Wertes
executeUpdate(stSql) für **INSERT**, **UPDATE** oder **DELETE** erwartet keine Rückgabe
execute(stSql) kann beliebig viele Rückgaben verarbeiten
Das ist in Zusammenhang mit der Firebird Datenbank wichtig, weil die zur Zeit (LO 7.4) so reagiert, dass z. B. Bei **ALTER**-Anweisungen eine (unverständliche) Rückmeldung erfolgt und damit die obige Prozedur scheinbar einen Fehler erzeugt, wenn sie mit **executeUpdate(stSql)** abläuft.

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

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

5 *https://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1sdbc_1_1XStatement.html*

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.[6]

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.

```
001 SUB Berechnung_ohne_MWSt(oEvent AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oFeld AS OBJECT
004    DIM oFeld2 AS OBJECT
005    oFeld = oEvent.Source.Model
006    oForm = oFeld.Parent
007    oFeld2 = oForm.getByName("Preis_ohne_MWSt")
008    oFeld2.BoundField.UpdateDouble(oFeld.getCurrentValue / 1.19)
009    IF NOT IsEmpty(oForm.getByName("Anzahl").getCurrentValue()) THEN
010       Berechnung_gesamt2(oForm.getByName("Anzahl"))
011    END IF
012 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.

```
001 SUB Berechnung_gesamt(oEvent AS OBJECT)
002    oFeld = oEvent.Source.Model
003    Berechnung_gesamt2(oFeld)
004 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.

```
001 SUB Berechnung_gesamt2(oFeld AS OBJECT)
002    DIM oForm AS OBJECT
003    DIM oFeld2 AS OBJECT
004    DIM oFeld3 AS OBJECT
005    DIM oFeld4 AS OBJECT
006    oForm = oFeld.Parent
007    oFeld2 = oForm.getByName("Preis")
008    oFeld3 = oForm.getByName("Preis_gesamt_mit_MWSt")
009    oFeld4 = oForm.getByName("MWSt_gesamt")
010    oFeld3.BoundField.UpdateDouble(oFeld.getCurrentValue * oFeld2.getCurrentValue)
011    oFeld4.BoundField.UpdateDouble(oFeld.getCurrentValue * oFeld2.getCurrentValue -
          oFeld.getCurrentValue * oFeld2.getCurrentValue / 1.19)
012 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 Steuer-

6 Siehe hierzu die Beispieldatenbank «Beispiel_Direktberechnung_im Formular.odb»

sätze 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:

```
001 FUNCTION OfficeVersion()
002    DIM aSettings, aConfigProvider
003    DIM aParams2(0) AS NEW com.sun.star.beans.PropertyValue
004    DIM sProvider$, sAccess$
005    sProvider = "com.sun.star.configuration.ConfigurationProvider"
006    sAccess = "com.sun.star.configuration.ConfigurationAccess"
007    aConfigProvider = createUnoService(sProvider)
008    aParams2(0).Name = "nodepath"
009    aParams2(0).Value = "/org.openoffice.Setup/Product"
010    aSettings = aConfigProvider.createInstanceWithArguments(sAccess, aParams2())
011    OfficeVersion() = array(aSettings.ooName,aSettings.ooSetupVersionAboutBox)
012 END FUNCTION
```

Diese Funktion gibt ein 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.

```
001 FUNCTION ID_Ermittlung(oFeld AS OBJECT) AS INTEGER
002    a() = OfficeVersion()
003    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
004       stInhalt = oFeld.currentValue
005    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.

```
006       stInhalt = oFeld.ValueItemList(oFeld.SelectedItems(0))
007    END IF
008    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.

```
009       ID_Ermittlung = -1
010    ELSE
011       ID_Ermittlung = Cint(stInhalt)
```

Der Text wird in eine Integer-Variable umgewandelt.

```
012    END IF
013 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:

```
001 SUB Listenfeldanzeige
002    DIM oDoc AS OBJECT
003    DIM oForm AS OBJECT
004    DIM oListbox AS OBJECT
005    DIM oController AS OBJECT
006    DIM oView AS OBJECT
007    oDoc = thisComponent
008    oForm = oDoc.Drawpage.Forms(0)
009    oListbox = oForm.getByName("Listenfeld")
010    oController = oDoc.getCurrentController()
011    oView = oController.getControl(oListbox)
012    print "Angezeigter Inhalt: " & oView.SelectedItem
013 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`**.

Auch der direkte Weg über die Liste der zur Anzeige zur Verfügung stehenden Inhalte ist möglich:

```
001 SUB ListView
002    oField = thisComponent.Drawpage.Forms(0).getByName("Listfield")
003    stView = oField.StringItemList(oField.SelectedItems(0))
004    msgbox stView
005 END SUB
```

Die Position des ausgewählten Werts steht in **`oField.SelectedItems(0)`**, da es sich bei Datenbanken um ein Listenfeld handelt, das keine Mehrfachselektion erlaubt. Mit der Position wird in der Liste der zur Anzeige zur Verfügung stehenden Inhalte ermittelt, welcher Inhalt an dieser Stelle steht. Beide Listen sind Arrays, so dass sie die entsprechende Zahlenangabe für die Position, beginnend mit 0, benötigen.

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 mit einem SQL-Befehl versehen, der eine Filterung unterschiedlicher Listenfeldabfragen vereinfacht[7]. Hier könnte z.B. stehen:

```
001 SELECT Tab".* FROM (SELECT "Name" AS "Field", "ID", "Name" AS "Sort" FROM
    "Tabelle") AS "Tab" ORDER BY "Field"
```

Dadurch wird klar, an welcher Stelle der Code später durch die Filterung ersetzt werden muss. Die Bedingungen der inneren Abfrage bleiben unberührt. Soll zuerst nur ein begrenzter Teil des gesamten Inhaltes angezeigt werden, so kann auch ein **`LIMIT`** für die Abfrage gesetzt werden. Durch die Zuweisung eines Alias in der Unterabfrage kann auf die Ermittlung von entsprechenden Feldnamen verzichtet werden. «Field» steht hier immer für das Feld, das gefiltert werden soll, «Sort» immer für die Sortierung. Das vereinfacht den Code für das Makro erheblich.

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

```
001 GLOBAL stListStart AS STRING
002 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.

7 Die Datenbank «Beispiel_Suchen_Filtern.odb» ist in den zusätzlichen Datenbanken mit besonderer Beschreibung jeder einzelnen Datenbank enthalten.

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.

```
001  SUB ListFilter(oEvent AS OBJECT)
002     oFeld = oEvent.Source.Model
003     IF oEvent.KeyCode < 538 OR oEvent.KeyCode = 1283 OR oEvent.KeyCode = 1284
004        OR oEvent.KeyCode = 1281 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 ← (Rücktaste). Wird dieser Code mit ausgelesen, so können auch Korrekturen durchgeführt werden. Mit dem **KeyCode** 1284 wird auch die Leertaste in die möglichen Zeichen aufgenommen. Hinter dem **KeyCode** 1281 steckt *ESC*. Die Taste soll zum Zurücksetzen des Listenfeldes dienen.

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.

```
005        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. Der Code wird direkt nach der Klammer für die Unterabfrage aufgetrennt. Die Unterabfrage hat hier den Alias **AS "Tab"**.

```
006        DIM stText AS STRING
007        DIM stFeld AS STRING
008        DIM stQuery AS STRING
009        DIM ar()
010        ar() = Split(oFeld.ListSource(0), " AS ""Tab""")
```

Für die Filterung der inneren Abfrage wird eine Bedingung erforderlich. An die bestehende Abfareg wird also zusammen mit dem Alias **WHERE** angehängt:

```
011        stQuery = ar(0) & " AS ""Tab"" WHERE"
012        IF oEvent.KeyCode = 1281 THEN  'Taste ESC
013           stListStart = "" 'Bei ESC wieder den gesamten Inhalt anzeigen
014        ELSEIF lZeit > 0 AND Timer() - lZeit < 5 THEN
015           stListStart = stListStart & oEvent.KeyChar
016        ELSE
017           stListStart = oEvent.KeyChar
018        END IF
019        lZeit = Timer()
```

Zuerst wird die Bedingung abgefragt, ob *ESC* gedrückt wurde. Hier kann nicht der **KeyChar** an die Abfrage weiter gegeben werden. Stattdessen muss ein leerer String weitergegeben werden, der die Suchergebnisse auf alle Daten ausdehnt. 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.

```
020        stText = LCase( stListStart & "%")
021        stSql(0) = stQuery+"LOWER(""Field"") LIKE '"+stText+"' ORDER BY ""Sort"""
022        oFeld.ListSource = stSql
```

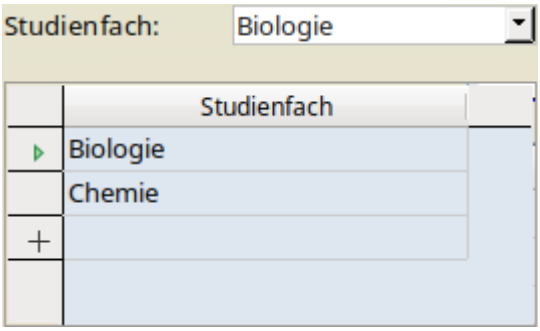
```
023         oFeld.refresh
024       END IF
025   END SUB
```

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.

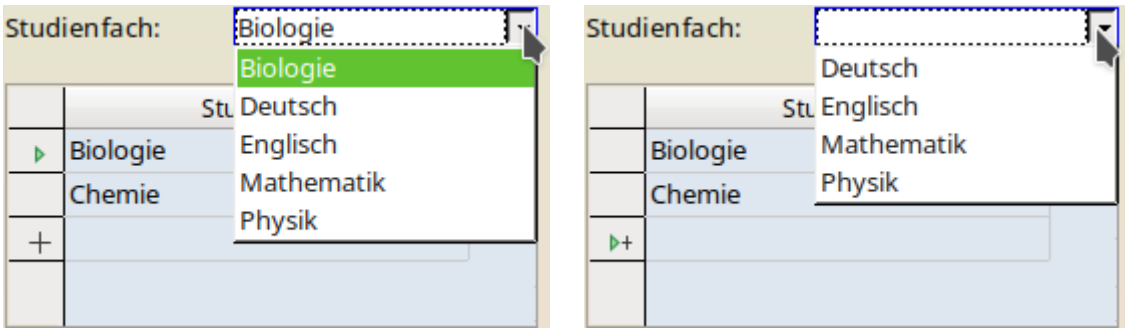
Soll eventuell nicht nach den Anfangsbuchstaben sondern ganz allgemein im angezeigten Inhalt gesucht werden, so muss lediglich die Variable «stText» von **`LCase( stListStart & "%")`** zu **`Lcase("%" & stListStart & "%")`** umgewandelt werden.

Listenfelder mit eingeschränkter Auswahl

Manchmal darf von einem Listenfeld nicht mehrmals der gleiche Wert ausgewählt werden. So sollte z. B. eine Person nicht doppelt einer Gruppe zugewiesen werden oder, wie im folgenden Screenshot, ein Studienfach mehrmals in der Liste der Studienfächer auftauchen.



In dem Screenshot wird das untere Tabellenkontrollfeld durch das darüberstehende Listenfeld mit Inhalten versehen. Ist das Listenfeld in dem Tabellenkontrollfeld eingebaut, so kann ein «Studienfach» mehrfach ausgewählt werden. Die Tabellenkonstruktion ist aber zur Sicherheit bereits so erstellt, dass der eingefügte Wert zusammen mit einem Wert des darüber liegenden Hauptformulars den Primärschlüssel bildet. Wird jetzt «Biologie» noch einmal gewählt, so hängt die Eingabe. Keine Rückmeldung, nur über *ESC* geht es wieder aus dieser Situation heraus.



Das Listenfeld zeigt jetzt nur die möglichen Einträge an. Das ist sowohl der aktuelle Datensatz als auch die anderen Fächer, die noch nicht ausgewählt wurden. Das Tabellenkontrollfeld kann nur zum Löschen von Datensätzen genutzt werden.

Wird ein neuer Datensatz bearbeitet, so stehen nur die zusätzlich verfügbaren Inhalte zur Auswahl. Die Fächer «Biologie» und «Chemie» erscheinen hier nicht.

Eine solche Konstruktion ist nicht innerhalb eines Tabellenkontrollfeldes möglich. Die Listenfelder werden hier nicht gleichbleibend aktualisiert, so dass zwischendurch die aktuell ausgewählten Fächer verschwinden würden, obwohl bereits ein entsprechender Fremdschlüssel abgespeichert wurde.

Der in dem Listenfeld stehende SQL-Code für die Anzeige der Liste sieht so aus:

```
001 SELECT "Tab".* FROM
002    (SELECT "Studienfach" AS "Field",
003    "ID" AS "Key",
004    "Studienfach" AS "Sort" FROM "tbl_Studienfach") AS "Tab"
005 ORDER BY "Sort"
```

In der Unterabfrage wird über ein Alias dem anzuzeigenden Inhalt, dem weiter zu gebendem Schlüsselwert und der beabsichtigten Sortierung ein fester Begriff zugeordnet. So kann auch bei komplizierterem Code über z. B. mehrere Tabellen immer Bezug auf diese Schlüsselpositionen genommen werden: **"Field"**, **"Key"** und **"Sort"**. Der eigentliche Code für das Listenfeld kann über den Alias für die Unterabfrage **AS "Tab"** von den über Makros hinzugefügten Bedingungen abgetrennt werden.

In den Zusatzinformationen zu dem Listenfeld steht ein Code, der in der Prozedur «Listenfeldfilter» zur Anzeige des zu dem aktuellen Datensatz passenden Listeninhaltes benötigt wird:

```
001 SELECT "Studienfach_ID" FROM "tbl_LehrerIn_Studienfach" WHERE "L_ID"
```

Die folgende Prozedur wird über **Formulareigenschaften → Ereignisse → Nach dem Datensatzwechsel** bei jedem neuen Datensatz wieder aufgerufen.

```
001 SUB Listenfeldfilter(oEvent AS OBJECT)
002    DIM oListField AS OBJECT
003    DIM oForm AS OBJECT
004    DIM stID AS STRING
005    DIM stIDCurrent AS STRING
006    DIM stNotIn AS STRING
007    DIM stSql(0) AS STRING
008    oForm = oEvent.Source
009    IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
010       IF NOT oForm.Parent.isBeforeFirst() AND NOT oForm.Parent.isAfterLast() THEN
011          stID = oForm.Parent.getByName("ID").BoundField.getString
012          IF stID <> "" THEN
013             oListField = oForm.getByname("lboStart")
014             stIDCurrent = oListField.BoundField.getString
015             IF stIDCurrent = "" THEN stIDCurrent = "-1"
016             stNotIn = oListField.Tag
017             stSource = oListField.ListSource(0)
018             ar() = Split(stSource, " AS ""Tab""")
019             stSource = ar(0) & " AS ""Tab"" WHERE ( ""Key"" NOT IN
                   ( " & stNotIn & " = '" & stID & "' )
                   OR ""Key"" = '" & stIDCurrent & "' ) ORDER BY ""Sort"""
020             stSql(0) = stSource
021             oListField.ListSource = stSql
022             oListField.refresh
023          END IF
024       END IF
025    END IF
026 END SUB
```

Der Code für die Listenfelder ist so vorgegeben, dass nur die unbekannten Werte nachgeliefert werden müssen. Dabei ist der Code für das Listenfeld nach dem Format SELECT * FROM (SELECT ...) erstellt.Die folgende Aliassetzungen werden vorgenommen: SELECT "Name" AS "Field", "ID" AS "Key", "Name" AS "Sort". Der Ergänzungscode beginnt mit AS "Tab". Der Alias ist notwendig, da bei machen Datenbanken (z. B. MariaDB) eine Unterabfrage nicht ohne Alias angenommen wird.

In Zeile 11 wird auf ein fest im übergeordneten Formular befindlichen Feld Bezug genommen. Hier wird extra die Bezeichnung der Formularfeldes genutzt, da der entsprechende Eintrag ja nicht unbedingt auch in der Datenquelle für das übergeordnete Formular mit dem Namen Verzeichnet ist. Nur wenn dieser Verbindungseintrag von Hauptformular zu Unterformular vorhanden ist wird der weitere Code des Makros ausgeführt.

Auch in Zeile 13 ist wieder ein Feld mit dem Feldnamen vorgegeben. Dies ist das Listenfeld, das zur Anzeige des Inhaltes dient. Es soll also immer die Bezeichnung «lboStart» erhalten.

Ist noch kein Eintrag für den aktuellen Datensatz erfolgt, so ist der aktuelle Wert des Listenfeldes leer. Hier wird dann stattdessen der Wert '-1' angenommen, der standardmäßig bei automatisch hoch zählenden Feldern nicht vorkommt.

Die Variable stNotIn speichert den Inhalt aus den Zusatzinformationen des Listenfeldes (Zeile 16). Anschließend wird der SQL-Code des Listenfeldes ausgelesen und so in ein Array eingelesen, dass das erste Element des Arrays den Startcode des Listenfeldes wiedergibt (Zeilen 17 und 18).

Schließlich wird der Code mit den entsprechenden Variablen neu zusammengefügt, so dass nur die Inhalte angeboten werden, die noch nicht ausgewählt wurden. Zusätzlich wird lediglich der Inhalt aufgenommen, der im aktuellen Datensatz bei einer vorhergehenden Auswahl bereits abgespeichert wurde. Ohne diese Ergänzung wäre das Listenfeld bei bestehenden Datensätzen ja grundsätzlich leer.

Der zusammengesetzte Inhalt sieht dann bei z. B. «ID = 1» und «stIDCurrent = 3» für die oben genannte Abfrage so aus:

```
001 SELECT "Tab".* FROM
002    (SELECT "Studienfach" AS "Field",
003    "ID" AS "Key",
004    "Studienfach" AS "Sort" FROM "tbl_Studienfach") AS "Tab"
005    WHERE "Key" NOT IN
006       (SELECT "Studienfach_ID" FROM "tbl_LehrerIn_Studienfach"
          WHERE "L_ID" = '1')
007       OR "Key" = '3'
008 ORDER BY "Sort"
```

Das Listenfeld sollte lediglich eine Prozedur erhalten, die zum Abspeichern des Inhaltes führt. Sie ist im Listenfeld über **Eigenschaften → Listenfeld → Ereignisse → Modifiziert** verknüpft.

```
001 SUB ListenfeldSpeichern(oEvent AS OBJECT)
002    oListField = oEvent.Source.Model
003    oForm = oListField.Parent
004    oListField.commit
005    IF oForm.IsNew THEN
006       oForm.insertRow
007    ELSE
008       oForm.updateRow
009    END IF
010    IF oForm.isLast THEN
011       oForm.moveToInsertRow
012    ELSE
013       oForm.next
014    END IF
015 END SUB
```

Über das auslösende Listenfeld wird auf das Formular zugegriffen. Der Inhalt des Listenfeldes wird in das Formular übertragen und abhängig davon, ob der Datensatz schon existiert oder neu ist abgespeichert.

Ab Zeile 10 wird lediglich der nächste Datensatz in dem Unterformular angesteuert. Solange Daten vorhanden sind kann dies mit «Next» geschehen. Steht der Datensatzzeiger aber auf dem letzten Datensatz, dann wird als nächster Datensatz ein neuer Datensatz eingefügt.

So kann das Listenfeld in dem obigen Beispiel durch mehrfache Betätigung ohne zusätzliche Bewegung der Maus dazu genutzt werden, alle erforderlichen Studienfächer auszuwählen.

Listenfelder zur Mehrfachauswahl nutzen

Die Nutzung der Mehrfachauswahl von Listenfeldern zeigt das folgende Formular[8]:

Die Mehrfachauswahl funktioniert zusammen mit Base nur über Makros. In dem obigen Formular werden zuerst die Termindaten eingetragen und abgespeichert, damit für das weitere Verfahren die Nummer des Primärschlüssels der Tabelle "Termine" über das Makro ausgelesen werden kann. Die Inhalte des darunter stehenden Listenfeldes werden beim Klick auf die entsprechenden Werte in Verbindung mit *Ctrl* oder *Alt* markiert und direkt gespeichert.

Die zugrunde liegende Datenbank hat den folgenden Aufbau:

Über die Makros muss die Tabelle "rel_Termine_Personen" beschrieben werden. Nach einem Wechsel des Datensatzes muss außerdem aus dieser Tabelle ausgelesen werden, auf welche Datensätze das Listenfeld einzustellen ist. Das Listenfeld kann dabei nicht mit dieser Tabelle verbunden werden, da in ein Datenbankfeld nur ein Wert eingetragen bzw. ausgelesen werden kann.

8 Die Datenbank «Beispiel_Listenfeld_Mehrfachauswahl.odb» ist den Beispieldatenbanken für dieses Handbuch beigefügt.

Die Prozedur «AuswahlSpeichern» wird in den Eigenschaften des Listenfeldes mit **Ereignisse → Modifiziert** verbunden.

```
001 SUB AuswahlSpeichern(oEvent AS OBJECT)
002    DIM oConnection AS OBJECT
003    DIM oSQL_Statement AS OBJECT
004    DIM oField AS OBJECT
005    DIM oForm AS OBJECT
006    DIM stID AS STRING
007    DIM i AS INTEGER
008    DIM k AS INTEGER
009    oField = oEvent.Source.Model
010    oForm = oField.Parent
011    stID = oForm.getString(oForm.FindColumn("ID")
```

Obwohl die im Feld "ID" gespeicherte Variable eine Integer-Variable ist, wird sie hier als String ausgelesen. Eine String-Variable kann besser für die folgende Bedingung als leer erkannt werden, da die Integer-Variable durch die vorherige Definition als Integer den Wert '0' annimmt, wenn in "ID" gar kein Wert steht.

```
012    IF stID <> "" THEN
013       oConnection = oForm.activeConnection()
014       oSQL_Statement = oConnection.createStatement()
```

Die Verbindung zur Datenbank wird über die Formularverbindung geholt, Das SQL-Statement wird vorbereitet. Danach werden einfach alle bisherigen Datensätze mit Personen, die zu dem Termin gehören, gelöscht.

```
015       FOR k = LBound(oField.ValueItemList()) TO UBound(oField.ValueItemList())
016          stSql = "DELETE FROM ""rel_Termine_Personen"" WHERE ""Termine_ID"" =
                '"+stID+"'"
017          oSQL_Statement.executeUpdate(stSql)
018       NEXT
```

Anschließend werden alle ausgewählten Werte neu in die Tabelle "rel_Termine_Personen" geschrieben

```
019       FOR i = LBound(oField.SelectedValues()) TO UBound(oField.SelectedValues())
020          stSql = "INSERT INTO ""rel_Termine_Personen"" (""Termine_ID"",
                ""Personen_ID"") VALUES ('"+stID+"','"+oField.SelectedValues(i)+"')"
021          oSQL_Statement.executeUpdate(stSql)
022       NEXT
023    END IF
024 END SUB
```

Die Prozedur «AuswahlErstellen» wird in den Eigenschaften des Formulars mit **Ereignisse → Nach dem Datensatzwechsel** verbunden.

```
001 SUB AuswahlErstellen(oEvent AS OBJECT)
002    DIM aFields()
003    DIM aValues()
004    DIM oConnection AS OBJECT
005    DIM oSQL_Statement AS OBJECT
006    DIM oResult AS OBJECT
007    DIM oForm AS OBJECT
008    DIM oField AS OBJECT
009    DIM stSql AS STRING
010    DIM stID AS STRING
011    DIM i AS INTEGER
012    DIM k AS INTEGER
013    oForm = oEvent.Source
014    oConnection = oForm.activeConnection()
015    oSQL_Statement = oConnection.createStatement()
016    aFields = Array("ListePersonen")
```

Die Listenfelder im Formular werden in ein Array geschrieben. Dies hat den Vorteil, dass gleich mehrere Listenfelder mit entsprechenden Inhalten gefüllt werden können.

```
017    stID = oForm.getString(oForm.FindColumn("ID")
018    IF stID <> "" THEN
```

Eine Schleife über alle Listenfelder wird gestartet. In diesem Fall läuft die Schleife ein einziges Mal durch.

```
019         FOR i = LBound(aFields()) TO UBound(aFields())
```

Der SQL-Code für die bestehende Auswahl der Personen zu dem betreffenden Termin wird erstellt.

```
020             stSql = "SELECT ""Personen_ID"" FROM ""rel_Termine_Personen"" WHERE
                   ""Termine_ID"" = '"+stID+"'"
```

Die Abfrage wird gestartet und das Ergebnis in ein Array lesen.

```
021             oResult = oSQL_Statement.executeQuery(stSql)
022             k = 0
023             WHILE oResult.Next
024                ReDim Preserve aValues(k)
025                aValues(k) = oResult.getString(1)
026                k = k + 1
027             WEND
```

Das Ergebnisarray wird in die Feldeigenschaften übernommen, so dass die Werte angezeigt werden. Dabei ist zu beachten, dass die Eigenschaft **`selectedValues()`** erst in LO-Versionen ab LO 4.1 und nicht unter AOO sowie älteren LO-Versionen bekannt ist. Hier müsste also gegebenenfalls nachgebessert werden.

```
028             oField = oForm.GetByName(aFields(i))
029             oField.selectedValues() = aValues()
030             NEXT
031       END IF
032   END SUB
```

Datumswert aus einem Formularwert in eine Datumsvariable umwandeln

```
001   FUNCTION Datumswert(oFeld AS OBJECT) AS DATE
002       a() = OfficeVersion()
003       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.

```
004          DIM stMonat AS STRING
005          DIM stTag AS STRING
006          stMonat = Right(Str(0) & Str(oFeld.CurrentValue.Month),2)
007          stTag =  Right(Str(0) & Str(oFeld.CurrentValue.Day),2)
008          Datumswert = CDateFromIso(oFeld.CurrentValue.Year & stMonat & stTag)
009       ELSE
010          Datumswert = CDateFromIso(oFeld.CurrentValue)
011       END IF
012   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.

Eingabemöglichkeiten in Feldern einschränken

Numerische Felder, formatierbare Felder und Datumsfelder lassen es zu, dass mit den Pfeiltasten die Werte geändert werden können. In einem Forum kam die Nachfrage, ob das nicht irgendwie unterbunden werden könnte. Folgendes Makro verhindert nicht die Änderung, macht sie aber direkt wieder rückgängig. Es muss an das Ereignis **Taste gedrückt** gebunden werden.

```
001  SUB KeyTest(oEvent AS OBJECT)
002     IF oEvent.KeyCode = 1025 OR oEvent.KeyCode = 1024 THEN
003        oEvent.Source.Model.reset()
004     ELSE
005        oEvent.Source.Model.commit
006     END IF
007  END SUB
```

Die beiden Pfeiltasten werden abgefangen und das Feld auf den Ausgangswert zurückgestellt. Bei der Betätigung jeder anderen Taste wird der Wert direkt festgeschrieben und ist nicht mehr über **reset()** erreichbar. Andernfalls würden alle nicht abgespeicherten Eingaben des Feldes mit den Pfeiltasten wieder aufgehoben.

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 die das Makro auch zugreifen kann.[9]

```
001  SUB Suche(stTabelle AS STRING)
002     DIM oDatenquelle AS OBJECT
003     DIM oVerbindung AS OBJECT
004     DIM oSQL_Anweisung AS OBJECT
005     DIM stSql AS STRING
006     DIM oAbfrageergebnis AS OBJECT
007     DIM oDoc AS OBJECT
008     DIM oDrawpage AS OBJECT
009     DIM oForm AS OBJECT
010     DIM oForm2 AS OBJECT
011     DIM oFeld AS OBJECT
012     DIM stInhalt AS STRING
013     DIM arInhalt() AS STRING
014     DIM inI AS INTEGER
015     DIM inK AS INTEGER
016     oDoc = thisComponent
017     oDrawpage = oDoc.drawpage
018     oForm = oDrawpage.forms.getByName("Suchform")
019     oFeld = oForm.getByName("Suchtext")
020     stInhalt = oFeld.getCurrentValue()
021     stInhalt = LCase(stInhalt)
```

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

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

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.

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

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

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

Hinweis

Der SQL-Code müsste für Firebird hier angepasst werden:

```
027 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.

```
029        InI = 0
030        WHILE oAbfrageergebnis.next
031           ReDim Preserve arInhalt(inI)
032           arInhalt(inI) = oAbfrageergebnis.getString(1)
033           inI = inI + 1
034        WEND
035        stSql = "DROP TABLE ""Suchtmp"" IF EXISTS"
036        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)`**).

```
037        stSql = "SELECT """+arInhalt(0)+"""INTO ""Suchtmp"" FROM """+stTabelle+"""
              WHERE "
038        FOR inK = 0 TO (inI - 1)
039           stSql = stSql+"LOWER("""+arInhalt(inK)+""") LIKE '%"+stInhalt+"%'"
```

```
040             IF inK < (inI - 1) THEN
041                stSql = stSql+" OR "
042             END IF
043          NEXT
044          oSQL_Anweisung.executeQuery(stSql)
045       ELSE
046          stSql = "DELETE FROM ""Suchtmp"""
047          oSQL_Anweisung.executeUpdate (stSql)
048       END IF
```

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

```
049       oForm2 = oDrawpage.forms.getByName("Anzeige")
050       oForm2.reload()
051    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:

```
001 SELECT * FROM "Suchtabelle"
002 WHERE "Nr." IN ( SELECT "Nr." FROM "Suchtmp" )
003    OR "Nr." =
004       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.

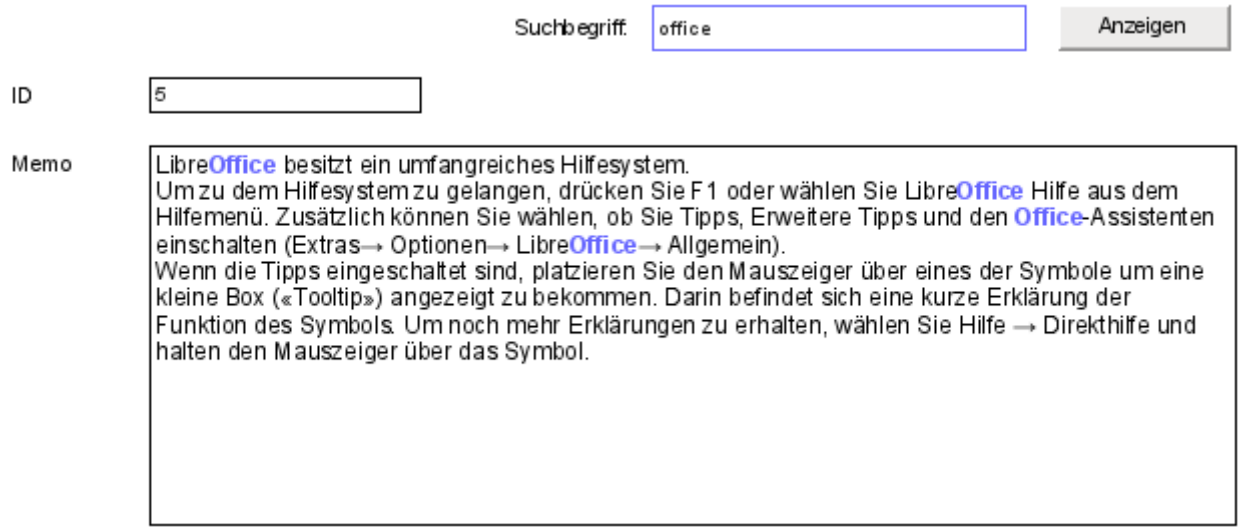
Bei größeren Datenmengen wird der Abgleich von Werten über die Verknüpfung IN aber zusehends langsamer. Es bietet sich also nicht an, für eine leere Eingabe in das Suchfeld einfach alle Primärschlüsselfelder der **"Suchtabelle"** in die Tabelle **"Suchtmp"** zu übertragen und dann auf die gleiche Art 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:

```
003    OR "Nr." =
004       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 bringen, bedarf es ein paar zusätzlicher Griffe in die Trickkiste.[10]

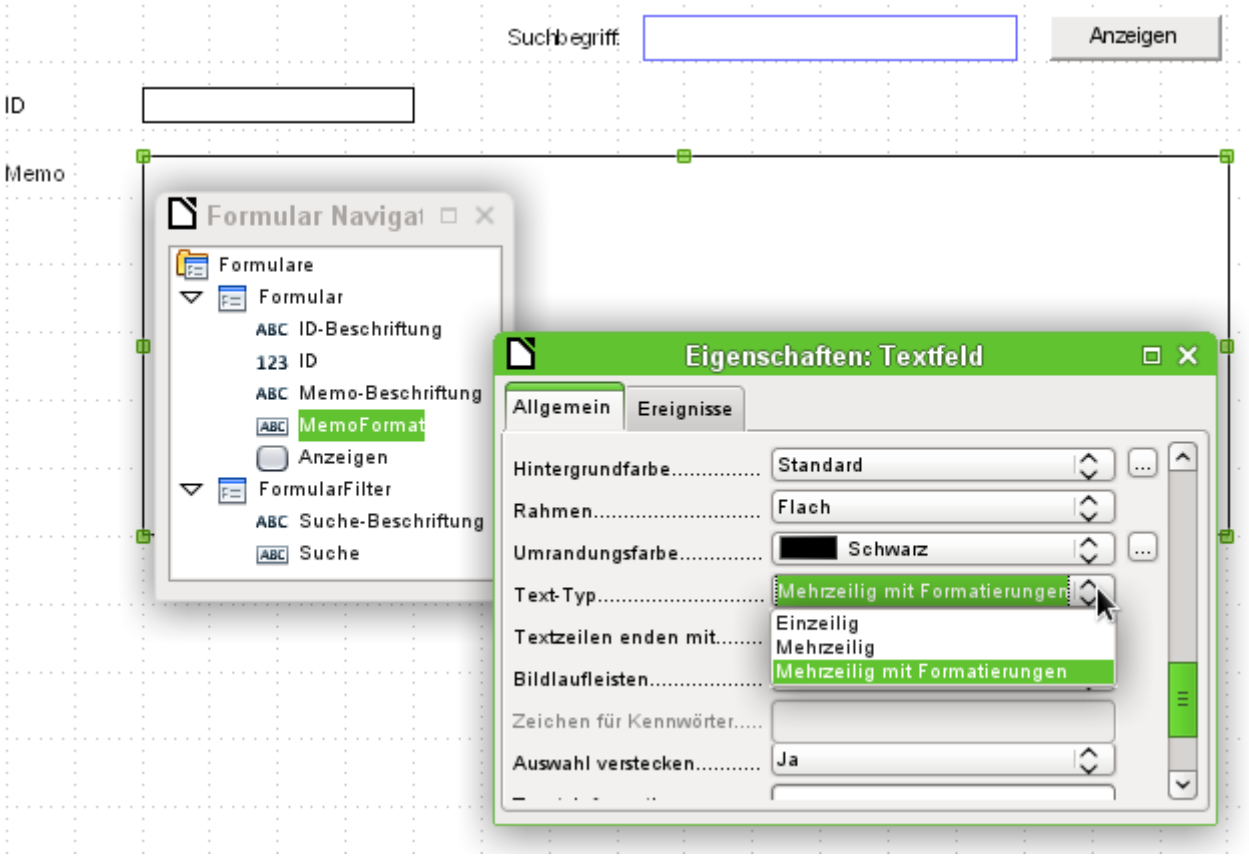
Die Funktionsweise eines solchen Suchfeldes wurde bereits bei den Abfragetechniken erklärt: Es wird eine Filtertabelle erstellt. Über ein Formular wird nur der aktuelle Wert des einzigen Datensatzes in dieser Tabelle neu geschrieben. Das Hauptformular wird über eine Abfrage mit dem entsprechenden Inhalt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

```
001 SELECT "ID", "Memo"
002 FROM "Tabelle"
003 WHERE LOWER ( "Memo" ) LIKE '%' || LOWER (
004    ( 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.

10 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**

```
001 Sub InhaltUebertragen(oEvent AS OBJECT)
002     DIM inMemo AS INTEGER
003     DIM oFeld AS OBJECT
004     DIM stSuchtext AS STRING
005     DIM oCursor AS OBJECT
006     DIM inSuch AS INTEGER
007     DIM inSuchAlt AS INTEGER
008     DIM inLen AS INTEGER
009     oForm = oEvent.Source
010     inMemo = oForm.findColumn("Memo")
011     oFeld = oForm.getByName("MemoFormat")
012     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.

```
013     Wait 5
014     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.

```
015     oCursor = oFeld.createTextCursor()
016     oCursor.gotoStart(true)
017     oCursor.gotoEnd(true)
018     oCursor.CharHeight = 10
019     oCursor.CharFontName = "Arial, Helvetica, Tahoma"
020     oCursor.CharColor = RGB(0,0,0)
021     oCursor.CharWeight = 100.000000   'com::sun::star::awt::FontWeight
022     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.

```
023     IF oFeld.Text <> "" AND stSuchtext <> "" THEN
024        IF inStr(oFeld.Text, stSuchtext) THEN
025           inSuch = 1
026           inSuchAlt = 0
027           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.

```
028           DO WHILE inStr(inSuch, oFeld.Text, stSuchtext) > 0
029              inSuch = inStr(inSuch, oFeld.Text, stSuchtext) - 1
030              oCursor.goRight(inSuch-inSuchAlt,false)
031              oCursor.goRight(inLen,true)
032              oCursor.CharColor = RGB(102,102,255)
033              oCursor.CharWeight = 110.000000
034              oCursor.goLeft(inLen,false)
035              inSuchAlt = inSuch
036              inSuch = inSuch + 2
037           LOOP
```

```
038         END IF
039      END IF
040   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**

```
001   Sub InhaltSchreiben(oEvent AS OBJECT)
002      DIM oForm AS OBJECT
003      DIM inMemo AS INTEGER
004      DIM loID AS LONG
005      DIM oFeld AS OBJECT
006      DIM stMemo AS STRING
007      oForm = oEvent.Source
008      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.

```
009         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».

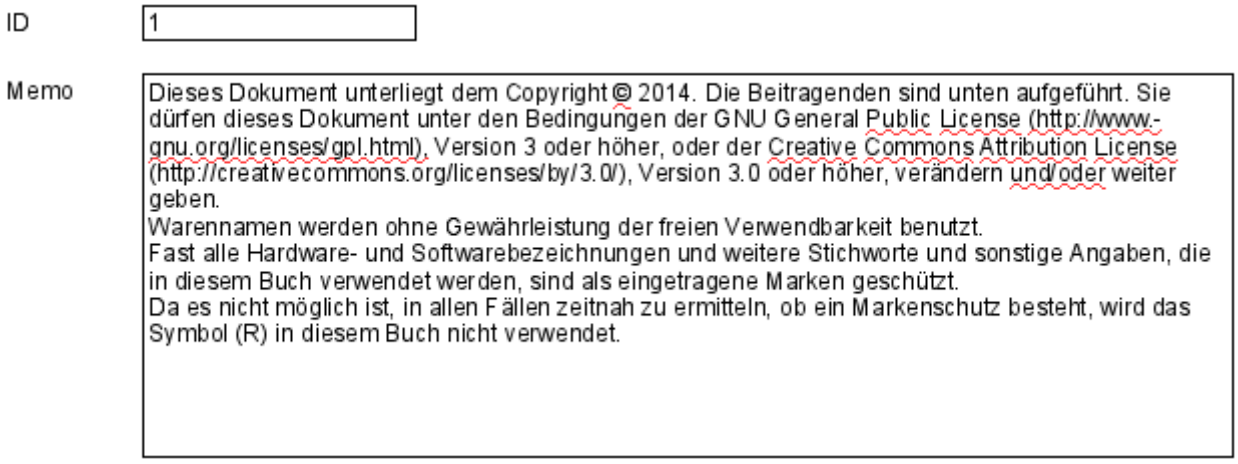
```
010            inMemo = oForm.findColumn("Memo")
011            loID = oForm.findColumn("ID")
012            oFeld = oForm.getByName("MemoFormat")
013            stMemo = oFeld.Text
014            IF stMemo <> "" THEN
015               oForm.updateString(inMemo,stMemo)
016            END IF
017            IF stMemo <> "" AND oForm.getString(loID) <> "" THEN
018               oForm.UpdateRow()
019            END IF
020         END IF
021      END IF
022   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.[11]

11 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**

```
001 SUB MarkierungFehlerDirekt(oEvent AS OBJECT)
002    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.

```
003    DIM aProp() AS NEW com.sun.star.beans.PropertyValue
004    DIM oLinuSvcMgr AS OBJECT
005    DIM oSpellChk AS OBJECT
006    DIM oFeld AS OBJECT
007    DIM arText()
008    DIM stWort AS STRING
009    DIM inlenWort AS INTEGER
010    DIM ink AS INTEGER
011    DIM i AS INTEGER
012    DIM oCursor AS OBJECT
013    DIM stText AS OBJECT
014    oLinguSvcMgr = createUnoService("com.sun.star.linguistic2.LinguServiceManager")
015    IF NOT IsNull(oLinguSvcMgr) THEN
016       oSpellChk = oLinguSvcMgr.getSpellChecker()
017    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.

```
018    oFeld = oEvent.Source.Model
019    ink = 0
020    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 ↵ (Eingabetaste) 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 ↵ (Eingabetaste) 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.

```
021         stText = Join(Split(oFeld.Text,CHR(10))," ")
022         stText = Join(Split(stText,CHR(13))," ")
023         arText = Split(RTrim(stText)," ")
024         FOR i = LBound(arText) TO Ubound(arText)
025            stWort = arText(i)
026            inlenWort = len(stWort)
027            stWort = Trim( RtrimStr( RtrimStr( RtrimStr( RtrimStr( RtrimStr(
                  RtrimStr(stWort,","), "."),"?"),"!"),"."),")"))
028            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.

```
029            IF stWort <> "" THEN
030               oCursor = oFeld.createTextCursor()
031               oCursor.gotoStart(false)
032               oCursor.goRight(ink,false)
033               oCursor.goRight(inLenWort,true)
034               If Not oSpellChk.isValid(stWort, "de", aProp()) Then
035                  oCursor.CharUnderline = 9
036                  oCursor.CharUnderlineHasColor = True
037                  oCursor.CharUnderlineColor = RGB(255,51,51)
038               ELSE
039                  oCursor.CharUnderline = 0
040               END IF
041            END IF
042            ink = ink + inLenWort + 1
043         NEXT
044      END IF
045   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.[12]

12 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.

```
001  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.

```
002     DIM oForm AS OBJECT
003     DIM oFeld AS OBJECT
004     DIM oFeldList AS OBJECT
005     DIM stAbfrage AS STRING
006     DIM stFeldWert AS STRING
007     DIM stFeldID AS STRING
008     DIM inCom AS INTEGER
009     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.

```
010     aComboboxen() = Split(oForm.getByName("Comboboxen").Tag,",")
011     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.

```
012         oFeldList = oForm.getByName(Trim(aComboboxen(inCom)))
013         stFeldID = oForm.getString(oForm.findColumn(oFeldList.Tag))
014         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 Sortieranweisung 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.

```
015         IF stFeldID <> "" THEN
016            stAbfrage = oFeldList.ListSource
017            IF InStr(stAbfrage,"order by") > 0 THEN
018               stSql = Left(stAbfrage, InStr(stAbfrage,"order by")-1)
019            ELSE
020               stSql = stAbfrage
021            END IF
022            IF InStr(stSql,"where") THEN
023               st = Right(stSql, Len(stSql)-InStr(stSql,"where")-4)
024               IF InStr(Left(st, InStr(st,"=")),"""ID""") THEN
025                  a() = Split(Right(st, Len(st)-InStr(st,"=")-1),".")
026               ELSE
027                  a() = Split(Left(st, InStr(st,"=")-1),".")
028               END IF
029               stSql = stSql + "AND "+a(0)+"""ID""" = "+stFeldID
030            ELSE
031               stSql = stSql + "WHERE ""ID""" = "+stFeldID
032            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.

```
033            oDatenquelle = ThisComponent.Parent.CurrentController
034            IF NOT (oDatenquelle.isConnected()) Then
035               oDatenquelle.connect()
036            End IF
037            oVerbindung = oDatenquelle.ActiveConnection()
038            oSQL_Anweisung = oVerbindung.createStatement()
039            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)**.

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

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

```
043          oFeldList.Text = stFeldWert
044       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.

```
045          oFeldList.Text = ""
046       END IF
047    NEXT inCom
048 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.

```
001 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.

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

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

```
003       oForm = oEvent.Source
004       aComboboxen() = Split(oForm.getByName("Comboboxen").Tag,",")
005       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.

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

```
010             oFeldList = oForm.getByName(a(0))
011          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:

```
001 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:

```
001 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:

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

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

```
012          stAbfrage = oFeldList.ListSource
013          aFelder() = Split(stAbfrage, """")
014          stInhalt = ""
015          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.

```
016             IF Trim(UCASE(aFelder(i))) = "ORDER BY" THEN
017                EXIT FOR
018             ELSEIF Trim(UCASE(aFelder(i))) = "FROM" THEN
019                stInhalt = stInhalt+" §§ "
020             ELSEIF Trim(UCASE(aFelder(i))) = "WHERE" THEN
021                stInhalt = stInhalt+" §§ "
022             ELSE
023                stInhalt = stInhalt+Trim(aFelder(i))
024             END IF
025          NEXT i
026          aInhalt() = Split(stInhalt, " §§ ")
```

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

```
027          aErster() = Split(aInhalt(0),"||")
028          IF UBound(aErster) > 0 THEN
029             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:

```
030                aTest() = Split(aErster(0),".")
031                NameTabelle1 = aTest(0)
032                NameTabellenFeld1 = aTest(1)
033                Erase aTest
034                Feldtrenner = Join(Split(aErster(1),"'"),"")
035                aTest() = Split(aErster(2),".")
036                NameTabelle2 = aTest(0)
037                NameTabellenFeld2 = aTest(1)
038                Erase aTest
039                aTest() = Split(aInhalt(2),"=")
040                aTest1() = Split(aTest(0),".")
041                IF aTest1(1) <> "ID" THEN
042                   NameTab12ID = aTest1(1)
043                   IF aTest1(0) = NameTabelle1 THEN
044                   Position = 2
045                   ELSE
046                      Position = 1
047                   END IF
048                ELSE
049                   Erase aTest1
050                   aTest1() = Split(aTest(1),".")
051                   NameTab12ID = aTest1(1)
052                   IF aTest1(0) = NameTabelle1 THEN
053                      Position = 2
054                   ELSE
055                      Position = 1
056                   END IF
057                END IF
058             ELSE
```

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

```
059                NameTabellenFeld1 = aErster(0)
060                Feldtrenner = Join(Split(aErster(1),"'"),"")
061                NameTabellenFeld2 = aErster(2)
062                NameTabelle1 = aInhalt(1)
063             END IF
064          ELSE
```

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

```
065             NameTabellenFeld1 = aErster(0)
066             NameTabelle1 = aInhalt(1)
067          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.

```
068          LaengeFeld1 = Spaltengroesse(NameTabelle1,NameTabellenFeld1)
069          IF NameTabellenFeld2 <> "" THEN
070             IF NameTabelle2 <> "" THEN
071                LaengeFeld2 = Spaltengroesse(NameTabelle2,NameTabellenFeld2)
072             ELSE
073                LaengeFeld2 = Spaltengroesse(NameTabelle1,NameTabellenFeld2)
074             END IF
075          ELSE
076             LaengeFeld2 = 0
077          END IF
```

Der Inhalt des Kombinationsfeldes wird ausgelesen:

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

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

```
079          stInhalt = Trim(stInhalt)
080          IF stInhalt <> "" THEN
081             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.

```
082                IF ASC(Feldtrenner) = 32 THEN
083                   Feldtrenner = " "
084                END IF
085                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.

```
086                IF Position = 2 OR Position = 0 THEN
087                   stInhalt = Trim(a_stTeile(0))
088                   IF UBound(a_stTeile()) > 0 THEN
089                      stInhaltFeld2 = Trim(a_stTeile(1))
090                   ELSE
091                      stInhaltFeld2 = ""
092                   END IF
093                   stInhaltFeld2 = Trim(a_stTeile(1))
094                ELSE
095                   stInhaltFeld2 = Trim(a_stTeile(0))
096                   IF UBound(a_stTeile()) > 0 THEN
097                      stInhalt = Trim(a_stTeile(1))
098                   ELSE
099                      stInhalt = ""
100                   END IF
101                   stInhalt = Trim(a_stTeile(1))
102                END IF
103             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.

```
104             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.

```
105                stmsgbox1 = "Das Feld " + NameTabellenFeld1 + " darf höchstens " +
                      LaengeFeld1 + "Zeichen lang sein." + CHR(13)
106                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.

```
107                IF (LaengeFeld1 > 0 AND Len(stInhalt) > LaengeFeld1) AND
                      (LaengeFeld2 > 0 AND Len(stInhaltFeld2) > LaengeFeld2) THEN
108                   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 optio-

nal 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.

```
109                ELSEIF (LaengeFeld1 > 0 AND Len(stInhalt) > LaengeFeld1) THEN
110                   msgbox ("Der eingegebene Text ist zu lang." + CHR(13) +
                         stmsgbox1 + "Bitte den Text kürzen.",64,"Fehlerhafte Eingabe")
111                ELSE
112                   msgbox ("Der eingegebene Text ist zu lang." + CHR(13) +
                         stmsgbox2 + "Bitte den Text kürzen.",64,"Fehlerhafte Eingabe")
113                END IF
114             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.

```
115                stInhalt = String_to_SQL(stInhalt)
116                IF stInhaltFeld2 <> "" THEN
117                   stInhaltFeld2 = String_to_SQL(stInhaltFeld2)
118                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.

```
119                inID1 = -1
120                inID2 = -1
121                oDatenquelle = ThisComponent.Parent.CurrentController
122                If NOT (oDatenquelle.isConnected()) Then
123                   oDatenquelle.connect()
124                End If
125                oVerbindung = oDatenquelle.ActiveConnection()
126                oSQL_Anweisung = oVerbindung.createStatement()
127                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.

```
128                   stSql = "SELECT ""ID"" FROM """ + NameTabelle2 + """ WHERE """ +
                         NameTabellenFeld2 + """='" + stInhaltFeld2 + "'"
129                   oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
130                   WHILE oAbfrageergebnis.next
131                      inID2 = oAbfrageergebnis.getInt(1)
132                   WEND
133                   IF inID2 = -1 THEN
134                      stSql = "INSERT INTO """ + NameTabelle2 + """ (""" +
                            NameTabellenFeld2 + """) VALUES ('" + stInhaltFeld2 + "') "
135                      oSQL_Anweisung.executeUpdate(stSql)
136                      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.

Hinweis

Die Funktion **`CALL IDENTITY()`** ist in ~~Firebird~~ unbekannt. Firebird hat hierfür eigentlich die Funktion **`RETURNING`** vorgesehen, die direkt an den SQL-Befehl angehängt wird und der Ausgabe den Primärschlüsselwert mitgibt. Leider funktioniert dies unter LO zur Zeit nicht. Stattdessen muss über eine separate Abfrage der gerade erzeugte Primärschlüsselwert ermittelt werden.

```
137                     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
138                     WHILE oAbfrageergebnis.next
139                           inID2 = oAbfrageergebnis.getInt(1)
140                     WEND
141                  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.

```
142                  stSql = "SELECT ""ID"" FROM """ + NameTabelle1 + """ WHERE """ +
                        NameTab12ID + """='" + inID2 + "' AND """ +
                        NameTabellenFeld1 + """ = '" + stInhalt + "'"
143                  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
144                  WHILE oAbfrageergebnis.next
145                     inID1 = oAbfrageergebnis.getInt(1)
146                  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.

```
147                  IF inID1 = -1 THEN
148                     stSql = "INSERT INTO """ + NameTabelle1 + """ (""" +
                           NameTabellenFeld1 + """,""" + NameTab12ID + """)
                           VALUES ('" + stInhalt + "','" + inID2 + "') "
149                     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.

```
150                     stSql = "CALL IDENTITY()"
151                     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
152                     WHILE oAbfrageergebnis.next
153                        inID1 = oAbfrageergebnis.getInt(1)
154                     WEND
155                  END IF
156               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:

```
157               IF NameTabellenFeld2 <> "" AND NameTabelle2 = "" THEN
158                  stSql = "SELECT ""ID"" FROM """ + NameTabelle1 + """ WHERE """ +
                        NameTabellenFeld1 + """='" + stInhalt + "' AND """ +
                        NameTabellenFeld2 + """='" + stInhaltFeld2 + "'"
159                  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
```

```
160                     WHILE oAbfrageergebnis.next
161                        inID1 = oAbfrageergebnis.getInt(1)
162                     WEND
163                     IF inID1 = -1 THEN
```

Wenn eine zweite Tabelle nicht existiert:

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

Anschließend wird das Primärschlüsselfeld wieder ausgelesen.

```
166                        stSql = "CALL IDENTITY()"
167                        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
168                        WHILE oAbfrageergebnis.next
169                              inID1 = oAbfrageergebnis.getInt(1)
170                        WEND
171                     END IF
172                  END IF
173                  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.

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

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

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

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

```
182                        stSql = "CALL IDENTITY()"
183                        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
184                        WHILE oAbfrageergebnis.next
185                              inID1 = oAbfrageergebnis.getInt(1)
186                        WEND
187                     END IF
188                  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 in StarBasic») diesem Feld zugewiesen.

```
189                  oForm.updateLong(oForm.findColumn(oFeldList.Tag),inID1)
190               END IF
191            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.

```
192               oForm.updateNULL(oForm.findColumn(oFeldList.Tag),NULL)
193            END IF
194        NEXT inCom
```

```
195     END IF
196  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.

```
001  FUNCTION Spaltengroesse(Tabellenname AS STRING, Feldname AS STRING) AS INTEGER
002     oDatenquelle = ThisComponent.Parent.CurrentController
003     If NOT (oDatenquelle.isConnected()) Then
004        oDatenquelle.connect()
005     End If
006     oVerbindung = oDatenquelle.ActiveConnection()
007     oSQL_Anweisung = oVerbindung.createStatement()
008     stSql = "SELECT ""COLUMN_SIZE"" FROM ""INFORMATION_SCHEMA"".""SYSTEM_COLUMNS""
           WHERE ""TABLE_NAME"" = '" + Tabellenname + "' AND ""COLUMN_NAME"" = '"
           + Feldname + "'"
009     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
010     WHILE oAbfrageergebnis.next
011        i = oAbfrageergebnis.getInt(1)
012     WEND
013     Spaltengroesse = i
014  END FUNCTION
```

Hinweis

Für Firebird muss der SQL-Code angepasst werden:

```
008  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

Dieses Makro sollte an das **Ereignis → Bei Fokuserhalt** des Listenfeldes gebunden werden. 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.

```
001  SUB Datensatzaktion_erzeugen(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     oForm = oEvent.Source.Model.Parent
004     oForm.IsModified = TRUE
005  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 **Formular-Eigenschaften → Zusatzinformationen** (Tag) der Name des Formulars eintragen. Hier können auch weitere Informationen eingetragen werden, die über den Befehl **`Split()`** anschließend voneinander getrennt werden.

```
001 SUB Zu_Formular_von_Formular(oEvent AS OBJECT)
002    DIM stTag AS String
003    stTag = oEvent.Source.Model.Tag
004    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.

```
005    ThisDatabaseDocument.FormDocuments.getByName( Trim(aForm(0)) ).open
006    IF Ubound(aForm()) > 0 THEN
007       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForm(1)) ).close
008    END IF
009 END SUB
```

Soll grundsätzlich das aktuelle Formular geschlossen werden, so braucht nur in den Zusatzinformationen des Buttons für das folgende Makro das Zielformular angegeben zu werden:

```
001 SUB Zu_Formular_von_Formular(oEvent AS OBJECT)
002    DIM stZiel AS String
003    aFormStart() = Split(thisComponent.Title, thisComponent.UntitledPrefix)
004    stZiel = oEvent.Source.Model.Tag
005    ThisDatabaseDocument.FormDocuments.getByName( Trim(stZiel) ).open
006    ThisDatabaseDocument.FormDocuments.getByName( Trim(aFormStart(1)) ).close
007 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 angesteuert werden, so ist das folgende Makro einfach an das Formular unter **Extras → Anpassen → Ereignisse → Dokument wird geschlossen** anzubinden:

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

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

```
001 SUB Zu_Formular_von_Formular_mit_Ordner(oEvent AS OBJECT)
002    REM Das zu öffenende Formular wird als erstes angegeben.
003    REM Liegt ein Formular in einem Ordner, so ist die Beziehung über "/" zu
004    REM definieren, so dass der Unterordner zu finden ist.
005    DIM stTag AS STRING
006    stTag = oEvent.Source.Model.Tag 'Tag wird unter den Zusatzinformationen
          eingegeben
007    aForms() = Split(stTag, ",")   'Hier steht zuerst der Formularname für das neue
          Formular, dann der für das alte Formular
008    aForms1() = Split(aForms(0),"/")
009    aForms2() = Split(aForms(1),"/")
010    IF UBound(aForms1()) = 0 THEN
011       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms1(0)) ).open
012    ELSE
013       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms1(0)) ).getByName(
             Trim(aForms1(1)) ).open
014    END IF
015    IF UBound(aForms2()) = 0 THEN
016       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms2(0)) ).close
017    ELSE
018       ThisDatabaseDocument.FormDocuments.getByName( Trim(aForms2(0)) ).getByName(
             Trim(aForms2(1)) ).close
019    END IF
020 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")`**.

Datensatz im Formular direkt öffnen

Wird von einem Formular zum anderen gesprungen, so kann das Zielformular natürlich über eine Filtertabelle mit einem bestimmten Schlüssel versehen werden und direkt nur mit einem Datensatz geöffnet werden. Manchmal ist es aber erforderlich, direkt eine Übersicht über mehrere Datensätze zu haben und dennoch den korrekten Datensatz direkt in einem Tabellenkontrollfeld angezeigt zu bekommen. Das im folgenden vorgestellte Makro springt bei entsprechender Vorgabe direkt zu dem gewünschten Datensatz.

```
001  SUB DatumsAenderung(oEvent AS OBJECT)
002     DIM oDoc AS OBJECT
003     DIM oDrawpage AS OBJECT
004     DIM oForm AS OBJECT
005     DIM oDatField AS OBJECT
006     DIM oConnection AS OBJECT
007     DIM oSQL_Statement AS OBJECT
008     DIM oResult AS OBJECT
009     DIM iRow AS INTEGER
010     DIM stDatum AS STRING
011     DIM stSql AS STRING
012     oDatField = oEvent.Source.Model
013     stDatum = oDatField.CurrentValue.Year & "-" &
           Right("0" & oDatField.CurrentValue.Month , 2) & "-" &
           Right("0" & oDatField.CurrentValue.Day , 2)
014     oDoc = thisComponent
015     oDrawpage = oDoc.drawpage
016     oForm = oDrawpage.forms.getByName("Uebersicht")
017     oConnection = oForm.activeConnection()
018     oSQL_Statement = oConnection.createStatement()
019     stSql = oForm.SingleSelectQueryComposer.Query
020     oResult = oSQL_Statement.executeQuery(stSql)
021     Do
022        oResult.next
023        iRow = iRow +1
024        IF oResult.isLast THEN Exit Do
025     LOOP UNTIL stDatum = oResult.getString(2)
026     oForm.last
027     oForm.absolute(iRow)
028  END SUB
```

Bei diesem Beispiel wird zuerst aus einem Formularfeld für ein Datum ein bestimmter Wert ausgelesen. Dieser Datumswert wird in die für SQL übliche Schreibweise umgewandelt. Das Formular soll auf den ersten Datensatz eingestellt werden, der mit dem Datumswert übereinstimmt.

Das Formular wird angesteuert. Für die Ermittlung der Position des Datensatzes ist es nun wichtig, genau zu wissen, mit welchem SQL-Kommando denn der Inhalt des Formulars gefüllt wurde. Dieses SQL-Kommando befindet sich nicht eindeutig in **`oForm.Command`**. Das ist lediglich das Kommando, das bei der Erstellung des Formulars ausgesucht wird. Im Formular selbst kann aber noch sortiert und gefiltert werden. Entweder müssten also zusätzlich zu dem Command noch die Filterung und Sortierung berücksichtigt werden oder nach einer fertigen Zusammenstellung in den Formulareigenschaften gesucht werden. Die fertige Zusammenstellung unter Berücksichtigung von Filter und Sortierung befindet sich in **`oForm.SingleSelectQueryComposer.Query`**. Dieser Composer steht nur dann zur Verfügung, wenn die Abfrage nicht im direkten SQL-Modus ausgeführt wird.

In der dem Formular zugrundeliegenden Abfrage wird jetzt nach dem Datum gesucht. Dazu wird mit **`DO ... LOOP UNTIL`** eine Schleife durchlaufen, die dann endet, wenn der in der Abfrage an 2. Position stehende Datumswert genau der Vorgabe entspricht, oder wenn der letzte Datensatz erreicht ist. Für jede neu ausgelesene Zeile wird der Zähler **`iRow`** um '1' erhöht.

Zum Schluss wird das Formular auf den letzten Datensatz eingestellt und von dort aus dann zurück auf die ermittelte Datenzeile. Das hat den Vorteil, dass das Datum in diesem Fall im

Tabellenkontrollfeld auf jeden Fall oben steht und außerdem noch klar ist, wie viele Datensätze denn im Moment über das Formular verfügbar sind.

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.

```
001 SUB Navigation(oEvent AS OBJECT)
002    DIM stTag AS STRING
003    DIM inType AS INTEGER
004    stTag = oEvent.Source.Model.Tag
005    aOpen() = Split(stTag, ",")
006    SELECT CASE Trim(aOpen(0))
007       CASE "form", "report"
008          REM Forms and Reports could be saved also in subfolders.
009          aForms1() = Split(Trim(aOpen(1)),"/")
010          IF Trim(aOpen(0)) = "form" THEN
011             oDoc = ThisDatabaseDocument.FormDocuments
012          ELSE
013             oDoc = ThisDatabaseDocument.ReportDocuments
014          END IF
015          IF Ubound(aForms1()) > 0 THEN
016             oDoc.getByName( Trim(aForms1(0)) ).getByName( Trim(aForms1(1)) ).open
017          ELSE
018             oDoc.getByName( Trim(aForms1(0)) ).open
019          END IF
020          IF Trim(aOpen(0)) = "form" AND Ubound(aOpen()) > 1 THEN
021             REM The Form, which starts the Macro, could also be closed ...
022             aForms2() = Split(Trim(aOpen(2)),"/")
023             IF Ubound(aForms2()) > 0 THEN
024                ThisDatabaseDocument.FormDocuments.
                      getByName( Trim(aForms2(0)) ).getByName( Trim(aForms2(1)) ).close
025             ELSE
026                ThisDatabaseDocument.FormDocuments.
                      getByName( Trim(aForms2(0)) ).close
027             END IF
028          END IF
029          EXIT SUB
030       CASE "query"
031          inType = 1
032          Open_Table_Query_View(Trim(aOpen(1)),inType)
033       CASE "table"
034          inType = 0
035          Open_Table_Query_View(Trim(aOpen(1)),inType)
036    END SELECT
037 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.

```
001  SUB Open_Table_Query_View(stName AS STRING, inType AS INTEGER)
002     DIM oController AS OBJECT
003     DIM oConnection AS OBJECT
004     oController = ThisDatabasedocument.CurrentController
005     IF NOT oController.isconnected THEN oController.connect
006     oConnection = oController.ActiveConnection
007     DIM URL AS NEW com.sun.star.util.URL
008     DIM Args(5) AS NEW com.sun.star.beans.PropertyValue
009     URL.Complete = ".component:DB/DataSourceBrowser"
010     Dispatch = StarDesktop.queryDispatch(URL,"_Blank",8)
011     Args(0).Name = "ActiveConnection"
012     Args(0).Value = oConnection
013     Args(1).Name = "CommandType"
014     Args(1).Value = inType   '0=Table 1=SQL_Query 2=Command
015     Args(2).Name = "Command"
016     Args(2).Value = stName
017     Args(3).Name = "ShowMenu"
018     Args(3).Value = True
019     Args(4).Name = "ShowTreeView"
020     Args(4).Value = False
021     Args(5).Name = "ShowTreeViewButton"
022     Args(5).Value = False
023     Dispatch.dispatch(URL, Args)
024  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.[13]

13 Siehe hierzu die Beispieldatenbank «Beispiel_Suchen_und_Filter.odb»

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

Filterung des Formulars mit hierarchischen Listenfeldern

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.

```
001 SUB Hierarchisches_Kontrollfeld(oEvent AS OBJECT)
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeldHidden AS OBJECT
006    DIM oFeld AS OBJECT
007    DIM oFeld1 AS OBJECT
008    DIM stSql AS STRING
009    DIM aInhalt()
010    DIM stTag AS STRING
011    oFeld = oEvent.Source.Model
012    stTag = oFeld.Tag
013    oForm = oFeld.Parent
014    REM Tag wird unter den Zusatzinformationen eingegeben
015    REM Hier steht:
016    REM 0. Feldname des zu filterndes Feldes in der Tabelle,
017    REM 1. Feldname des versteckten Konrollfeldes, das den Filterwert speichern
          soll,
018    REM 2. eventuell weiteres Listfeld
019    REM Der Tag wird von dem auslösenden Element ausgelesen. Die Variable wird an
          die Prozedur weitergegeben, die gegebenenfalls alle weiteren Listenfelder
          einstellt.
020    aFilter() = Split(stTag, ",")
021    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.

```
022    IF Trim(aFilter(1)) = "" THEN
023       IF oFeld.getCurrentValue <> "" THEN
024          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.

```
025          IF oForm.Filter <> ""
                AND InStr(oForm.Filter, """"+Trim(aFilter(0))+"""='") = 0 THEN
026             stFilter = oForm.Filter + " AND " + stFilter
```

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.

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.

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

```
030             END IF
031         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.

```
032         oForm.Filter = stFilter
033         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.

```
034     ELSE
035         oFeldHidden = oForm.getByName(Trim(aFilter(1)))
036         IF oFeld.getCurrentValue <>"" THEN
037             stFilter = """"+Trim(aFilter(0))+"""='"+oFeld.getCurrentValue()+"'"
038             IF oFeldHidden.HiddenValue <> ""
                    AND InStr(oFeldHidden.HiddenValue, """"+Trim(aFilter(0))+"""='") = 0
                    THEN
039                 stFilter = oFeldHidden.HiddenValue + " AND " + stFilter
040             ELSEIF oFeldHidden.HiddenValue <> "" THEN
041                 stFilter = Left(oFeldHidden.HiddenValue,
                        InStr(oFeldHidden.HiddenValue, """"+Trim(aFilter(0))+"""='")-1) +
                        stFilter
042             END IF
043         END IF
044         oFeldHidden.HiddenValue = stFilter
045     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.

```
046     IF UBound(aFilter()) > 1 THEN
047         oFeld1 = oForm.getByName(Trim(aFilter(2)))
048         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.

```
049         IF oFeld.getCurrentValue <> "" THEN
050             stSql = "SELECT DISTINCT """+Trim(aFilter1(0))+""" FROM """+oForm.Command+
                    """ WHERE "+stFilter+" ORDER BY """+Trim(aFilter1(0))+""""
051             oDatenquelle = ThisComponent.Parent.CurrentController
052             If NOT (oDatenquelle.isConnected()) THEN
053                 oDatenquelle.connect()
054             END IF
055             oVerbindung = oDatenquelle.ActiveConnection()
056             oSQL_Anweisung = oVerbindung.createStatement()
057             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.

```
058             inZaehler = 0
059             WHILE oAbfrageergebnis.next
060                 ReDim Preserve aInhalt(inZaehler)
061                 aInhalt(inZaehler) = oAbfrageergebnis.getString(1)
062                 inZaehler = inZaehler+1
063             WEND
064         ELSE
065             aInhalt(0) = ""
```

```
066         END IF
067         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.

```
068         oFeld1.refresh()
069         WHILE UBound(aFilter1()) > 1
070            DIM aLeer()
071            oFeld2 = oForm.getByName(Trim(aFilter1(2)))
072            DIM aFilter1()
073            aFilter1() = Split(oFeld2.Tag,",")
074            oFeld2.StringItemList = aLeer()
075            oFeld2.refresh()
076         WEND
077      END IF
078   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.

```
001      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.

Hierarchische Listenfelder in der Formulareingabe nutzen

Auch bei der Eingabe von Formularen können solche hierarchischen Listenfelder genutzt werden. Die hier aufgeführten Makros erledigen dabei nur die notwendigen Grundlagen[14]. Die Felder für das 3. Listenfeld werden z.B. nicht automatisch zurückgestellt, wenn aus dem ersten Listenfeld ein neuer Wert ausgesucht wird.

Das Makro ist an **Eigenschaften: Listenfeld → Ereignisse → Vor dem Aktualisieren** gebunden. Diese Eigenschaft steht bei Listenfeldern auch innerhalb von Tabellenkontrollfeldern zur Verfügung. So ist das Makro sowohl bei Tabellenkontrollfeldern als auch bei einfachen Formularfeldern universell nutzbar.

```
001   SUB Listenfeldfilter(oEvent AS OBJECT)
002      DIM stSql(0) AS STRING
003      DIM oForm AS OBJECT
004      DIM oFeld AS OBJECT
005      DIM oFeld2 AS OBJECT
006      DIM stFeld AS STRING
007      DIM stWert AS STRING
008      DIM stFeld2 AS STRING
009      DIM stSqlFeld2 AS STRING
010      oFeld = oEvent.Source
011      stFeld = oFeld.DataField
012      stWert = oFeld.CurrentValue
013      oForm = oFeld.Parent ' oForm ist bei Tabellenkontrollfelder das Tabellenobjekt
```

14 Siehe hierzu die Beispieldatenbank «Beispiel_hierarchische_Listenfelder.odb»

```
014     stFeld2 = oFeld.Tag
015     oFeld2 = oForm.getByname(stFeld2)
016     stSqlFeld2 = oFeld2.ListSource(0)
```

In dem auslösenden Feld steht lediglich in den Zusatzinformationen der Name des Formularfeldes, das neu eingestellt werden soll.

Aus dem auslösenden Feld wird das Datenfeld ausgelesen. Sollte hier in der Datenbank für das Feld eine andere Bezeichnung gewählt worden sein als für die Abfrage im folgenden Listenfeld notwendig, so muss dies ebenfalls in den Zusatzinformationen aufgeführt werden.

Aus dem auslösenden Feld wird auch der momentane Wert ausgelesen. Das ist seit der Version LO 4.1 der Wert, der tatsächlich in der Datenbank abgespeichert wird.

Das Zielfeld wird angesteuert und der Inhalt der dort enthaltenen Abfrage über **`ListSource(0)`** ausgelesen. Da es sein kann, dass durch eine vorherige Betätigung des auslösenden Feldes hier bereits eine **`WHERE`**-Bedingung steht, muss diese gegebenenfalls in der folgenden Schleife entfernt werden.

```
017     IF InStr(stSqlFeld2, "WHERE") THEN
018        ar = Split(stSqlFeld2, "WHERE")
019        stSqlFeld2 = ar(0)
020     END IF
```

Zum Schluss wird der Code als der erste Wert des Arrays stSql zusammengestellt. Er wird als **`ListSource`** an das Zielfeld übergeben. Das Feld wird mit einem **`refresh`** auf den neuen Inhalt eingestellt.

```
021     stSql(0) = stSqlFeld2 & " WHERE """+stFeld+""" = '"+stWert+"'"
022     oFeld2.ListSource = stSql
023     oFeld2.refresh
024 END SUB
```

Wir so ein Makro für Listenfelder in einem **Tabellenkontrollfeld** genutzt, so muss **Formulareigenschaften → Daten → Daten ändern → 'Nein'** ausgewählt sein. Sonst werden die Listenfelder in den vorhergehenden Formularfeldern geändert und können dort gegebenenfalls die alten Daten nicht mehr anzeigen.

Die folgenden beiden Makros nutzen eine Filtertabelle. Das hat den Vorteil, dass in den Listenfeldern beliebiger SQL-Code stehen kann. Die Felder müssen lediglich in dem SQL-Code einen Verweis auf den Tabellenwert stehen haben. Das obere Beispiel funktioniert so wie aufgeschrieben hingegen nur, wenn der SQL-Code direkt nach der Tabellenbenennung endet und keine **`WHERE`**-Bedingung und keine Sortierung enthält.

```
001 SUB Listfeldfilter_Tabelle(oEvent AS OBJECT)
002     DIM oDatasource AS OBJECT
003     DIM oConnection AS OBJECT
004     DIM oSQL_Statement AS OBJECT
005     DIM oForm AS OBJECT
006     DIM oFeld AS OBJECT
007     DIM oFeld2 AS OBJECT
008     DIM stFeld AS STRING
009     DIM stWert AS STRING
010     DIM stSql AS STRING
011     oFeld = oEvent.Source
012     stFeld = oFeld.DataField
013     stWert = oFeld.CurrentValue
014     oForm = oFeld.Parent ' oForm ist bei Tabellenkontrollfelder das Tabellenobjekt
015     stFeld2 = oFeld.Tag
016     oFeld2 = oForm.getByname(stFeld2)
017     oDatasource = thisDatabaseDocument.CurrentController
018     IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
019     oConnection = oDatasource.ActiveConnection()
020     oSQL_Statement = oConnection.createStatement()
021     stSql ="UPDATE ""Filter"" SET """+stFeld+""" = '"+stWert+"' WHERE ""ID"" = TRUE"
022     oSQL_Statement.executeUpdate(stSql)
```

```
023     oFeld2.refresh
024 END SUB
```

Die Prozedur «Listfeldfilter_Tabelle» startet wie die vorhergehende Prozedur. Der SQL-Code des Zielfeldes spielt hier gar keine Rolle. Er muss lediglich so gestaltet sein, dass er durch Werte in der Tabelle "Filter" beeinflusst wird. Hier der Beispielcode für das Listenfeld, das die Klasse ausgeben soll:

```
001 SELECT "Klasse", "ID" FROM "Klasse"
002 WHERE "J_ID" =
        COALESCE ( ( SELECT "J_ID" FROM "Filter" WHERE "ID" = TRUE ), "J_ID" )
003 ORDER BY "Klasse" ASC
```

In dem Makro wird jetzt schlicht z.B. der Inhalt des Feldes "J_ID" aus der Tabelle "Filter" neu beschrieben und das Ziellistenfeld neu eingelesen.

Der Reset des Filters ist notwendig, damit bei der nächsten Eingabe wieder alle Werte in den Listenfeldern vorhanden sind. Er wird deshalb an die **Formulareigenschaften → Ereignisse → Vor dem Datensatzwechsel** gebunden.

```
001 SUB Filter_Reset(oEvent AS OBJECT)
002     DIM oConnection AS OBJECT
003     DIM oSQL_Statement AS OBJECT
004     DIM oForm AS OBJECT
005     DIM stSql AS STRING
006     oForm = oEvent.Source
007     IF inStr(oForm.ImplementationName,"ODatabaseForm") THEN
008        oConnection = oForm.activeConnection()
009        oSQL_Statement = oConnection.createStatement()
010        stSql = "UPDATE ""Filter"" SET ""J_ID"" = NULL, ""K_ID"" = NULL
              WHERE ""ID"" = TRUE"
011        oSQL_Statement.executeUpdate(stSql)
012     END IF
013 END SUB
```

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 **Formulareigenschaften → Ereignisse → Vor der Datensatzaktion** gebunden wird:

```
001 SUB Timestamp
002     DIM unoStmp AS NEW com.sun.star.util.DateTime
003     DIM oDoc AS OBJECT
004     DIM oDrawpage AS OBJECT
005     DIM oForm AS OBJECT
006     DIM oFeld AS OBJECT
007     DIM stZeit AS STRING
008     DIM ar()
009     DIM arMandS()
010     DIM loNano AS LONG
011     DIM inSecond AS INTEGER
012     DIM inMinute AS INTEGER
013     oDoc = thisComponent
014     oDrawpage = oDoc.Drawpage
015     oForm = oDrawpage.Forms.getByName("MainForm")
016     oFeld = oForm.getByName("Zeit")
017     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.

```
018     IF stZeit <> "" THEN
019        ar() = Split(stZeit,",")
020        loNano = CLng(ar(1)&"0000000")
021        arMandS() = Split(ar(0),":")
022        inSecond = Cint(arMandS(1))
023        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.

```
024        WITH unoStmp
025           .NanoSeconds = loNano
026           .Seconds = inSecond
027           .Minutes = inMinute
028           .Hours = 0
029           .Day = 30
030           .Month = 12
031           .Year = 1899
032        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:

```
001  DIM Jetzt AS DATE
002  Jetzt = Now()
003  WITH unoStmp
004     .NanoSeconds = loNano
005     .Seconds = inSecond
006     .Minutes = inMinute
007     .Hours = Hour(Jetzt)
008     .Day = Day(Jetzt)
009     .Month = Month(Jetzt)
010     .Year = Year(Jetzt)
011  END WITH
```

```
033        oFeld.BoundField.updateTimestamp(unoStmp)
034     END IF
035  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:

```
001  SUB Ereignisursache_ermitteln(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     oForm = oEvent.Source
```

```
004     MsgBox oForm.ImplementationName
005 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.

Sicherer soll hier die Abfrage sein, ob eine der Implementationen ein bestimmtes **UnoInterface** benutzt. Das ist fest in der API verankert und wird damit wohl nicht geändert:

```
001 IF hasUnoInterfaces(oForm, "com.sun.star.form.XForm" ) THEN
```

weist darauf hin, dass es sich um die Implementation **com.sun.star.comp.forms.ODatabaseForm** handelt. Auch die Ermittlung des **ServiceName** kann eine klare Abgrenzung bewirken. Die **SupportedServiceNames** sind in einem Array in **oForm** enthalten. Über

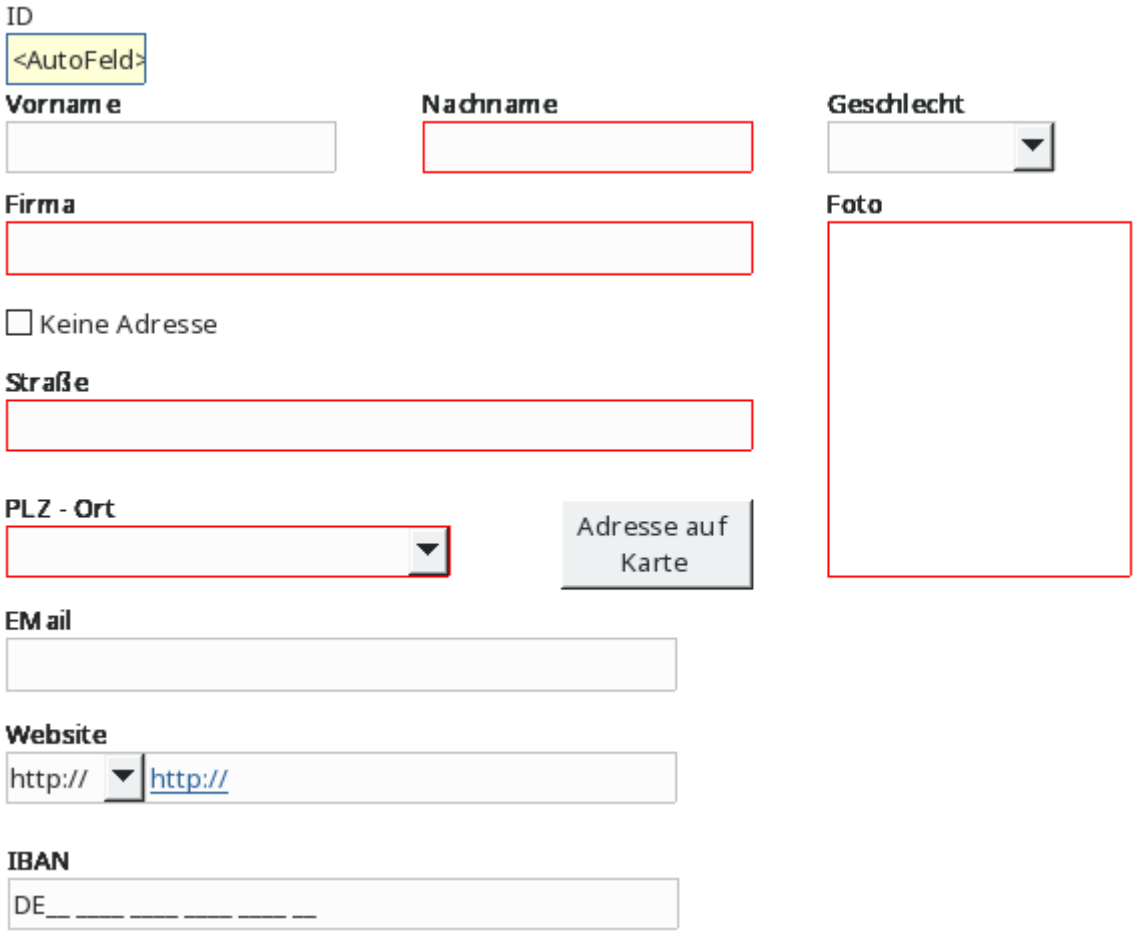
```
001 IF oForm.supportsService("com.sun.star.form.component.DataForm") THEN
```

kann hier **ODatabaseForm** ermittelt werden.

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 allerdings auch in der Tabelle der Datenbank definiert sein. Die diesem Abschnitt zugrundeliegende Datenbank[15] zeigt fehlende Eingabe direkt an und vermeidet in einigen Feldern auch eventuell fehlerhafte Eingaben.

15 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.

```
001  GLOBAL loBorderDefault AS LONG
002  GLOBAL loBorderInputRequired AS LONG
003  GLOBAL loColorStandard AS LONG
004  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.

```
001  SUB FormVars(oEvent AS OBJECT)
002     DIM oForm AS OBJECT, oField AS OBJECT
003     DIM k AS INTEGER, i AS INTEGER
004     loBorderDefault = RGB(192,192,192) 'Grau
005     loBorderInputRequired = RGB(255,0,0) 'Rot
006     loColorStandard = RGB(250,250,250) 'sehr helles Grau
007     oForm = oEvent.Source
008     FOR i = 0 TO oForm.Count - 1
009        oField = oForm.getByIndex(i)
010        IF oField.supportsService("com.sun.star.form.DataAwareControlModel") THEN
011           IF oField.InputRequired THEN
012              IF oField.getPropertySetInfo.hasPropertyByName("BorderColor") THEN
013                 REDIM PRESERVE arFormInputRequired(k)
014                 arFormInputRequired(k) = oField.Name
015                 k = k + 1
016              END IF
017           END IF
018        END IF
019     NEXT
020     FormChange(oEvent)
021  END SUB
```

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

```
001  SUB FormChange(oEvent AS OBJECT)
002     DIM oForm AS OBJECT, oField AS OBJECT
003     DIM i AS INTEGER, n AS INTEGER, k AS INTEGER
004     DIM stTest AS STRING
005     DIM a(), aa(), ab()
006     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.

```
007     FOR i = LBound(arFormInputRequired()) TO UBound(arFormInputRequired())
008        oField = oForm.getByName(arFormInputRequired(i))
009        oField.InputRequired = True
010        IF NOT IsNULL(oField.BoundField) THEN
011           IF oField.BoundField.String = "" THEN
012              oField.BorderColor = loBorderInputRequired
013           ELSE
014              oField.BorderColor = loBorderDefault
015           END IF
016        ELSEIF oField.CurrentValue = "" THEN
017           oField.BorderColor = loBorderInputRequired
018        ELSE
019           oField.BorderColor = loBorderDefault
020        END IF
021     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.

```
022     FOR i = LBound(arFormInputRequired()) TO UBound(arFormInputRequired())
023        oField = oForm.getByName(arFormInputRequired(i))
024        IF NOT IsNULL(oField.BoundField) THEN
025           stTest = oField.BoundField.String
026        ELSE
027           stTest = oField.CurrentValue
028        END IF
029        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:

```
030           a = split(oField.Tag,"]")
031           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.

```
032              IF InStr(a(n),"notrequired") THEN
033                 aa = split(a(n),"[")
034                 ab = split(aa(1),",")
035                 FOR k = LBound(ab()) TO UBound(ab())
036                    oField = oForm.getByName(ab(k))
037                    oField.BorderColor = loBorderDefault
038                    oField.InputRequired = False
039                 NEXT
040              END IF
041           NEXT
042        END IF
```

```
043     NEXT
044 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 wird bei **Eingabe erforderlich → Ja** gesetzt. Entsprechend werden auch die Rahmenfarben angepasst.

```
001 SUB NotRequired(oEvent AS OBJECT)
002     DIM oFieldStart AS OBJECT, oForm AS OBJECT
003     DIM n AS INTEGER, k AS INTEGER
004     DIM a(), aa(), ab()
005     oFieldStart = oEvent.Source.Model
006     oForm = oFieldStart.Parent
007     a = split(oFieldStart.Tag,"]")
008     FOR n = LBound(a()) TO UBound(a())-1
009        IF InStr(a(n),"notrequired") THEN
010           aa = split(a(n),"[")
011           ab = split(aa(1),",")
012           FOR k = LBound(ab()) TO UBound(ab())
013              oField = oForm.getByName(ab(k))
014              IF oFieldStart.CurrentValue <> "" THEN
015                 oField.BorderColor = loBorderDefault
016                 oField.InputRequired = False
017              ELSE
018                 oField.BorderColor = loBorderInputRequired
019                 oField.InputRequired = True
020              END IF
021           NEXT
022        END IF
023     NEXT
024 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.

```
001 SUB EnableDisable(oEvent AS OBJECT)
002     DIM oForm AS OBJECT, oField AS OBJECT
003     DIM stTag AS STRING
004     DIM i AS INTEGER, k AS INTEGER
005     DIM a(), aa(), ab()
006     oForm = oEvent.Source.Model.Parent
007     stTag = oEvent.Source.Model.Tag
008     a = split(stTag,"]")
009     FOR i = LBound(a()) TO UBound(a())-1
010        IF InStr(a(i),"inaktiv") THEN
011           aa = split(a(i),"[")
012           ab = split(aa(1),",")
013           FOR k = LBound(ab()) TO UBound(ab())
014              oField = oForm.getByName(ab(k))
015              IF oEvent.Source.Model.State THEN
016                 oField.Enabled = False
017                 oField.BorderColor = loBorderDefault
018                 oField.BackgroundColor = loBorderDefault
019              ELSE
020                 oField.Enabled = True
021                 oField.BorderColor = loBorderInputRequired
022                 oField.BackgroundColor = loColorStandard
023              END IF
```

```
024             NEXT
025          END IF
026       NEXT
027    END SUB
```

Die Prozedur «FieldRequired» wird an die Felder gebunden, bei denen die Eingabe zu Beginn auf 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.

```
001  SUB FieldRequired(oEvent AS OBJECT)
002     DIM oField AS OBJECT
003     oField = oEvent.Source.Model
004     IF oField.CurrentValue <> "" THEN
005        oField.BorderColor = loBorderDefault
006     ELSE
007        oField.BorderColor = loBorderInputRequired
008     END IF
009     IF inStr(oField.Tag,"notrequired") THEN
010        NotRequired(oEvent)
011     END IF
012  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.

```
001  SUB IBANValid(oEvent AS OBJECT)
002     DIM oField AS OBJECT, oForm AS OBJECT, oController AS OBJECT, oView AS OBJECT
003     DIM stMsg AS STRING, stText AS STRING, stLand AS STRING, stPruef AS STRING
004     DIM i AS INTEGER
005     DIM a()
006     oField = oEvent.Source.Model
007     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.

```
008     IF stText <> "" THEN
009        IF inStr(stText,"_") THEN
010           IF Val(Mid(stText,3)) > 0 THEN
011              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.

```
012           ELSE
013              oField.reset
014           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.

```
015         ELSE
016            a = split(stText)
017            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 noch 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.

```
018            i = cLng(a(1) & a(2)) Mod 97
019            i = cLng(i & a(3)) Mod 97
020            i = cLng(i & a(4)) Mod 97
021            i = cLng(i & a(5)) Mod 97
022            i = cLng(i & stLand) Mod 97
023            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.

```
024               stPruef = "131400"
025               i = cLng(a(1) & a(2)) Mod 97
026               i = cLng(i & a(3)) Mod 97
027               i = cLng(i & a(4)) Mod 97
028               i = cLng(i & a(5)) Mod 97
029               i = cLng(i & stPruef) Mod 97
030               i = 98 - i
031               stMsg "Die IBAN ist fehlerhaft." & CHR(13) & "Die Prüfziffer müsste "
032                  & i & " sein."
033            END IF
034         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.

```
035         IF stMsg <> "" THEN
036            msgbox (stMsg, 0, "Eingabe fehlerhaft")
037            oForm = oField.Parent
038            oController = thisComponent.getCurrentController()
039            oView = oController.getControl(oForm.getByname(oField.Name))
040            oView.setFocus
041         END IF
042      END IF
043   END SUB
```

Abspeichern nach erfolgter Kontrolle

Die Felder, bei denen eine Eingabe notwendig ist, 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.

```
001 FUNCTION SaveRequired(oEvent AS OBJECT) AS BOOLEAN
002    DIM oForm AS OBJECT, oField AS OBJECT
003    DIM stLabel AS STRING
004    DIM k AS INTEGER, i AS INTEGER
005    SaveRequired = True
006    oForm = oEvent.Source
007    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. Für «oDatabaseForm» muss die Funktion grundsätzlich **`True`** wiedergeben. In dem «FormController» sind die einzelnen Felder nur über das Model erreichbar.

```
008       FOR i = 0 TO oForm.Model.Count - 1
009          oField = oForm.Model.getByIndex(i)
010          IF oField.supportsService("com.sun.star.form.DataAwareControlModel") THEN
011             IF oField.InputRequired AND NOT IsNULL(oField.BoundField) THEN
012                IF oField.BoundField.String = "" THEN
013                   stLabel = stLabel & ", " & oField.LabelControl.Label
014                   k = k + 1
015                END IF
016             ELSEIF oField.InputRequired THEN
017                IF oField.CurrentValue = "" THEN
018                   stLabel = stLabel & ", " & oField.LabelControl.Label
019                   k = k + 1
020                END IF
021             ELSE
022             END IF
023          END IF
024       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.

```
025       IF k > 0 THEN
026          stLabel = Right(stLabel, Len(stLabel)-2)
027          IF k = 1 Then
028             stText = "Für das Feld "
029          ELSE
030             stText = "Für die Felder "
031          END IF
032          MsgBox ("Alle rot umrandeten Felder müssen einen Inhalt aufweisen." &
                CHR(13) & stText & stLabel & " ist eine Eingabe erforderlich." ,
                0 ,"Speichern gestoppt")
033          SaveRequired = False
034       END IF
035    END IF
036 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 Primärschlüssel existiert. So tauchen dann in der Tabelle als Primärschlüssel z.B. die folgenden Werte auf:[16]

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.

```
001 SUB Datum_aktuell_ID_einfuegen
002    DIM oDatenquelle AS OBJECT
003    DIM oVerbindung AS OBJECT
004    DIM oSQL_Anweisung AS OBJECT
005    DIM stSql AS STRING
006    DIM oAbfrageergebnis AS OBJECT
007    DIM oDoc AS OBJECT
008    DIM oDrawpage AS OBJECT
009    DIM oForm AS OBJECT
010    DIM oFeld1 AS OBJECT
011    DIM oFeld2 AS OBJECT
012    DIM oFeld3 AS OBJECT
013    DIM inIDneu AS INTEGER
014    DIM inYear AS INTEGER
015    DIM unoDate
016    oDoc = thisComponent
017    oDrawpage = oDoc.drawpage
018    oForm = oDrawpage.forms.getByName("MainForm")
019    oFeld1 = oForm.getByName("fmtJahr")
020    oFeld2 = oForm.getByName("fmtID")
021    oFeld3 = oForm.getByName("datDatum")
022    IF IsEmpty(oFeld2.getCurrentValue()) THEN
023       IF IsEmpty(oFeld3.getCurrentValue()) THEN
024          unoDate = createUnoStruct("com.sun.star.util.Date")
025          unoDate.Year = Year(Date)
026          unoDate.Month = Month(Date)
027          unoDate.Day = Day(Date)
028          inYear = Year(Date)
029       ELSE
030          inYear = oFeld3.CurrentValue.Year
031       END IF
032       oDatenquelle = ThisComponent.Parent.CurrentController
033       If NOT (oDatenquelle.isConnected()) THEN
034          oDatenquelle.connect()
035       END IF
036       oVerbindung = oDatenquelle.ActiveConnection()
037       oSQL_Anweisung = oVerbindung.createStatement()
038       stSql = "SELECT MAX( ""ID"" )+1 FROM ""Auftraege"" WHERE ""Jahr"" = '"
             + inYear + "'"
```

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

```
039         oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
040         WHILE oAbfrageergebnis.next
041            inIDneu = oAbfrageergebnis.getInt(1)
042         WEND
043         IF inIDneu = 0 THEN
044            inIDneu = 1
045         END IF
046         oFeld1.BoundField.updateInt(inYear)
047         oFeld2.BoundField.updateInt(inIDneu)
048         IF IsEmpty(oFeld3.getCurrentValue()) THEN
049            oFeld3.BoundField.updateDate(unoDate)
050         END IF
051      END IF
052   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 Formularfeld «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 Formularfeld «fmtID» eingetragen werden. Die Numerierung 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

```
001  oDatasource = ThisComponent.Parent.DataSource
002  IF NOT oDatasource.IsPasswordRequired THEN
003     oConnection = oDatasource.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:

```
004  ELSE
005     oAuthentication = createUnoService("com.sun.star.sdb.InteractionHandler")
006     oConnection = oDatasource.ConnectWithCompletion(oAuthentication)
007  END IF
```

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

```
oDatasource.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:

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

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

```
001  oDatasource = ThisComponent.Parent.CurrentController
002  IF NOT (oDatasource.isConnected()) Then
003     oDatasource.connect()
004  End IF
005  oConnection = oDatasource.ActiveConnection()
```

Die Datenbank ist hier bekannt, ein Nutzername und ein Passwort sind nicht erforderlich, da diese bereits in den Grundeinstellungen von Base für die internen Datenbankversionen ausgeschaltet sind.

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

```
001  oDatasource = ThisComponent.Drawpage.Forms(0)
002  oConnection = oDatasource.ActiveConnection
```

Hinweis

Es ist auch möglich, eine Datenbankverbindung ohne eine vorliegende Datenbankdatei zu erzeugen. Dies dürfte dann sinnvoll sein, wenn nur einzelne Informationen aus einer Datenquelle ausgelesen werden sollen und so etwas wie abgespeicherte Abfragen, Formulare und Berichte nicht benötigt werden.
Siehe hierzu die Ausführungen von Andrew Pitonyak in *https://www.pitonyak.org/database/AndrewBase.pdf* . Im Kapitel «Connections without a data source» ab S. 91 wird hier eine Verbindung über

```
001  oManager = CreateUnoService("com.sun.star.sdbc.DriverManager")
```

beschrieben. Diese Verbindung wird bei den Beispielen in diesem Handbuch bisher nicht weiter genutzt.

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.[17]

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.

```
001  SUB Datenkopie
002     DIM oDatabaseContext AS OBJECT
003     DIM oDatenquelle AS OBJECT
004     DIM oDatenquelleZiel AS OBJECT
005     DIM oVerbindung AS OBJECT
006     DIM oVerbindungZiel AS OBJECT
007     DIM oDB AS OBJECT
008     DIM oSQL_Anweisung AS OBJECT
009     DIM oSQL_AnweisungZiel AS OBJECT
010     DIM oAbfrageergebnis AS OBJECT
011     DIM oAbfrageergebnisZiel AS OBJECT
012     DIM stSql AS String
013     DIM stSqlZiel AS String
014     DIM inID AS INTEGER
015     DIM inIDZiel AS INTEGER
```

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

```
016     DIM stName AS STRING
017     DIM stOrt AS STRING
018     oDB = ThisComponent.Parent
019     stDir = Left(oDB.Location,Len(oDB.Location)-Len(ConvertToURL(oDB.Title))+8)
020     stDir = ConvertToUrl(stDir & "ZielDB.odb")
021     oDatenquelle = ThisComponent.Parent.CurrentController
022     If NOT (oDatenquelle.isConnected()) THEN
023        oDatenquelle.connect()
024     END IF
025     oVerbindung = oDatenquelle.ActiveConnection()
026     oDatabaseContext = createUnoService("com.sun.star.sdb.DatabaseContext")
027     oDatenquelleZiel = oDatabaseContext.getByName(stDir)
028     oVerbindungZiel = oDatenquelleZiel.GetConnection("","")
029     oSQL_Anweisung = oVerbindung.createStatement()
030     stSql = "SELECT * FROM ""Tabelle"""
031     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
032     WHILE oAbfrageergebnis.next
033        inID = oAbfrageergebnis.getInt(1)
034        stName = oAbfrageergebnis.getString(2)
035        stOrt = oAbfrageergebnis.getString(3)
036        oSQL_AnweisungZiel = oVerbindungZiel.createStatement()
037        stSqlZiel = "SELECT ""ID"" FROM ""Tabelle"" WHERE ""ID"" = '"+inID+"'"
038        oAbfrageergebnisZiel = oSQL_AnweisungZiel.executeQuery(stSqlZiel)
039        inIDZiel = - 1
040        WHILE oAbfrageergebnisZiel.next
041           inIDZiel = oAbfrageergebnisZiel.getInt(1)
042        WEND
043        IF inIDZiel = - 1 THEN
044           stSqlZiel = "INSERT INTO ""Tabelle"" (""ID"",""Name"",""Ort"") VALUES
              ('"+inID+"','"+stName+"','"+stOrt+"')"
045           oSQL_AnweisungZiel.executeUpdate(stSqlZiel)
046        END IF
047     WEND
048  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.[18]

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.

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

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.

```
001 Sub CalcDataImport(oEvent AS OBJECT, stBaseTab AS STRING, stCalcTab AS STRING)
002    DIM oDatasource AS OBJECT
003    DIM oConnection AS OBJECT
004    DIM oSQL_Command AS OBJECT
005    DIM oResult AS OBJECT
006    DIM oDB AS OBJECT
007    DIM oDoc AS OBJECT
008    DIM oDocView AS OBJECT
009    DIM oRange AS OBJECT
010    DIM Arg()
011    DIM aColumn()
012    DIM aColumns()
013    DIM aType()
014    DIM stSql AS STRING
015    DIM stRow AS STRING
016    DIM stDir AS STRING
017    DIM stColumns AS STRING
018    DIM stStartCol AS STRING
019    DIM stEndCol AS STRING
020    DIM stStartRow AS STRING
021    DIM stEndRow AS STRING
022    DIM stRange AS STRING
023    DIM inCounter AS INTEGER
024    DIM loColumns AS LONG
025    DIM loRows AS LONG
026    DIM loID AS LONG
027    oDatasource = ThisComponent.Parent.CurrentController
028    If NOT (oDatasource.isConnected()) THEN
029       oDatasource.connect()
030    END IF
031    oConnection = oDatasource.ActiveConnection()
032    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.

```
033    stSql = "SELECT COLUMN_NAME, TYPE_NAME FROM INFORMATION_SCHEMA.SYSTEM_COLUMNS
          WHERE TABLE_NAME = '"+stBaseTab+"' AND NOT COLUMN_NAME = 'ID'"
034    oResult = oSQL_Command.executeQuery(stSql)
035    inCounter = 0
036    stColumns = ""
```

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

Hinweis

Für Firebird muss der SQL-Code angepasst werden. Vor allem die Zuordnung der Datentypen in SQL-Format ist umständlicher:

```
033  stSql = "SELECT TRIM(""a"".RDB$FIELD_NAME), TRIM(CASE
          ""b"".RDB$FIELD_TYPE||'|'||
          COALESCE(""b"".RDB$FIELD_SUB_TYPE,0) "+ _
          "WHEN '7|0' THEN 'SMALLINT' "+ _
          "WHEN '8|0' THEN 'INTEGER' "+ _
          "WHEN '8|1' THEN 'NUMERIC' "+ _
          "WHEN '8|2' THEN 'DECIMAL' "+ _
          "WHEN '10|0' THEN 'FLOAT' "+ _
          "WHEN '12|0' THEN 'DATE' "+ _
          "WHEN '13|0' THEN 'TIME' "+ _
          "WHEN '14|0' THEN 'CHAR' "+ _
          "WHEN '16|0' THEN 'BIGINT' "+ _
          "WHEN '35|0' THEN 'TIMESTAMP' "+ _
          "WHEN '37|0' THEN 'VARCHAR' "+ _
          "WHEN '261|0' THEN 'BLOB' "+ _
          "WHEN '261|1' THEN 'BLOB Text' "+ _
          "WHEN '261|2' THEN 'BLOB BLR' "+ _
          "WHEN '261|3' THEN 'BLOB ACL' "+ _
          "END) AS ""SQL_Datentyp"" "+ _
          "FROM RDB$RELATION_FIELDS AS ""a"", RDB$FIELDS AS ""b"" "+ _
          "WHERE ""a"".RDB$FIELD_SOURCE = ""b"".RDB$FIELD_NAME "+ _
          "AND ""a"".RDB$RELATION_NAME = '"+stBaseTab+"' "+ _
          "AND ""a"".RDB$FIELD_NAME <> 'ID'"
```

Die Felder müssen außerdem mit **TRIM** eingeschränkt werden, da Firebird die Ausgabe mit fester Zeichenlänge macht und einfach Leerzeichen hinten anfügt. Für die Auswertung im Makro und den Vergleich ist das unpraktikabel.

```
037     WHILE oResult.next
038        ReDim Preserve aColumn(inCounter)
039        ReDim Preserve aType(inCounter)
040        aColumn(inCounter) = oResult.getString(1)
041        aType(inCounter) = oResult.getString(2)
042        inCounter = inCounter+1
043     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.

```
044     stSql = "SELECT MAX(""ID"") FROM """+stBaseTab+""""
045     oResult = oSQL_Command.executeQuery(stSql)
046     WHILE oResult.next
047        loID = oResult.getInt(1) + 1
048     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.

```
049     oDB = ThisComponent.Parent
050     stDir = Left(oDB.Location,Len(oDB.Location)-Len(ConvertToURL(oDB.Title))+8)
051     stDir = ConvertToUrl(stDir & "Daten_Calc.ods")
052     oDoc = StarDesktop.loadComponentFromURL(stDir,"_blank", 0, Arg() )
053     oDocView = oDoc.CurrentController.Frame.ContainerWindow
054     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.

```
055     loColumns = uBound(oDoc.Sheets.getByName(stCalcTab).ColumnDescriptions)
           ' Anzahl der Spalten
056     stStartCol = split(oDoc.Sheets.getByName(stCalcTab).ColumnDescriptions(0))(1)
057     stEndCol = split(oDoc.Sheets.getByName(stCalcTab).
           ColumnDescriptions(loColumns))(1)
058     loRows = uBound(oDoc.Sheets.getByName(stCalcTab).RowDescriptions)
           ' Anzahl der Zeilen
059     stStartRow = split(oDoc.Sheets.getByName(stCalcTab).RowDescriptions(0))(1)
060     stEndRow = split(oDoc.Sheets.getByName(stCalcTab).RowDescriptions(loRows))(1)
061     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.

```
062     oRange = oDoc.Sheets.getByName(stCalcTab).getCellRangeByName(stRange)
063     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**.

```
064     aColumns = aDat(0)
065     FOR i = 1 TO uBound(aDat)
066        aRow = aDat(i)
067        stColumns = """ID"""
068        stRow = loID
069        FOR k = 0 TO loColumns
070           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.

```
071              IF aColumns(k) = aColumn(n) THEN
072                 IF aType(n) = "DECIMAL" THEN
073                    IF aRow(k) <> "" THEN
074                       aRow(k) = Str(CDbl(aRow(k)))
075                    END IF
076                 END IF
077                 IF aType(n) = "DATE" THEN
078                    IF isDate(CDate(aRow(k))) AND aRow(k) <> "" THEN
079                       aRow(k) = Year(CDate(aRow(k))) & "-"
                             & Right("0" & Month(CDate(aRow(k))), 2) &"-"
                             & Right("0" & Day(CDate(aRow(k))), 2)
080                    ELSE
081                       aRow(k) = ""
082                    END IF
083                 END IF
084                 IF aType(n) = "TIME" THEN
085                    IF isDate(CDate(aRow(k))) AND aRow(k) <> "" THEN
```

```
086                     aRow(k) = Right("0" & Hour(CDate(aRow(k))), 2) & ":"
                           & Right("0" & Minute(CDate(aRow(k))), 2) & ":"
                           & Right("0" & Second(CDate(aRow(k))), 2)
087                  ELSE
088                     aRow(k) = ""
089                  END IF
090               END IF
091               IF aType(n) = "TIMESTAMP"
092                  IF isDate(CDate(aRow(k))) AND aRow(k) <> "" THEN
093                     aRow(k) = Year(CDate(aRow(k))) & "-"
                           & Right("0" & Month(CDate(aRow(k))), 2) & "-"
                           & Right("0" & Day(CDate(aRow(k))), 2) & " "
                           & Right("0" & Hour(CDate(aRow(k))), 2) & ":"
                           & Right("0" & Minute(CDate(aRow(k))), 2) & ":"
                           & Right("0" & Second(CDate(aRow(k))), 2)
094                  ELSE
095                     aRow(k) = ""
096                  END IF
097               END IF
```

Ist der Zellinhalt leer oder für ein Dezimalfeld, Datumsfeld, Zeitfeld oder Timestampfeld 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.

```
098               IF aRow(k) = "" THEN
099                  aRow(k) = "NULL"
100               ELSE
101                  aRow(k) = "'" & aRow(k) & "'"
102               END IF
103               stRow = stRow & "," & aRow(k)
104               stColumns = stColumns & ",""" & aColumns(k) & """"
105            END IF
106         NEXT
107      NEXT
108      stSql = "INSERT INTO """+stBaseTab+""" ("& stColumns &")
            VALUES ("& stRow &")"
109      oSQL_Command.executeUpdate(stSql)
```

Der Primärschlüsselwert wird hier innerhalb des Makros um 1 heraufgesetzt.

```
110      loID = loID + 1
111   NEXT
112   oDoc.close(True) ' Schließen des Calc-Documentes
113   oEvent.Source.Model.parent.reload()   ' Neuladen des Formulars
114 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.

Die Prozedur wird über eine andere Prozedur gestartet. Dadurch kann die Prozedur auch für mehrere Tabellen der Calc-Datei bzw. der Base-Datei genutzt werden:

```
001 SUB Import1_Start(oEvent AS OBJECT)
002    CalcDataImport(oEvent, "tbl_Dez_Dat_NULL", "Tab_Dez_Dat_NULL")
003 END SUB
```

Das auslösende Ereignis wird einfach weiter gereicht. Als zweiter Parameter wird der Tabellenname der Base-Tabelle angegeben. Als dritter Parameter erfolgt die Angabe des Tabellennamens aus der Calc-Datei.

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.

```
001 SUB Abfrageninhalt
002    DIM oDatenDatei AS OBJECT
003    DIM oAbfragen AS OBJECT
004    DIM stQuery AS STRING
005    oDatenDatei = ThisComponent.Parent.CurrentController.DataSource
006    oAbfragen = oDatenDatei.getQueryDefinitions()
007    stQuery = oAbfragen.getByName("Query").Command
008    msgbox stQuery
009 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 SQL-Codes 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.

Soll eine Abfrage in einem Formular weiter genutzt werden, so geht dies über die **COMMAND**-Variable des Formulars:

```
008 oForm.Command = stQuery
009 oForm.CommandType = com.sun.star.sdb.CommandType.COMMAND
010 oForm.reload()
```

Auch die Änderung von Abfragen über ein Makro ist über die Abfragedefinition möglich:

```
007 oAbfrage = oAbfragen.getByName("Query")
008 oAbfrage.Command = "SELECT …"
```

Mit Hilfe solch eine Konstruktion kann z. B. anschließend ein Bericht gestartet werden, der sich auf die Abfrage bezieht. Dadurch kann eine entsprechende Filterung der Daten vorgenommen werden ohne dass eine separate Filtertabelle erstellt werden muss. Dies ist vor allem dann von Vorteil, wenn es sich bei der zugrundeliegenden Datenbank nicht um eine Datenbank handelt, die Abfragen über mehrere Tabellen zulässt. Dies ist beispielsweise bei dBase der Fall. dBase-Tabellen lassen sich nicht in einer Abfrage kombinieren, über Makros dann aber sehr wohl entsprechend filtern, indem der Wert einer Makroabfrage anschließend dem Code der bestehenden Abfrage hinzugefügt wird.

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. Existiert das Verzeichnis noch

nicht, so wird es erstellt. Das Makro beginnt nach einer voreingestellten Zahl an Kopien (**inMax**) damit, die jeweils älteste Variante zu überschreiben.

Die Prozedur «Datenbankbackup» wird unter **Extras → Anpassen → Ereignisse → Dokument öffnen** eingebunden.

```
001 SUB Datenbankbackup(inMax AS INTEGER)
002    DIM oPath AS OBJECT
003    DIM oDoc AS OBJECT
004    DIM sTitel AS STRING
005    DIM sUrl_Ziel AS STRING
006    DIM sUrl_Start AS STRING
007    DIM i AS INTEGER
008    DIM k AS INTEGER
009    oDoc = ThisComponent
010    sTitel = oDoc.Title
011    sUrl_Start = oDoc.URL
012    DO WHILE sUrl_Start = ""
013       oDoc = oDoc.Parent
014       sTitel = oDoc.Title
015       sUrl_Start = oDoc.URL
016    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.

```
017    oPath = createUnoService("com.sun.star.util.PathSettings")
018    MkDir(oPath.Backup)
019    FOR i = 1 TO inMax + 1
020       IF NOT FileExists(oPath.Backup & "/" & i & "_" & sTitel) THEN
021          IF i > inMax THEN
022             FOR k = inMax - 1 TO 1 STEP -1
023                IF FileDateTime(oPath.Backup & "/" & k & "_" & sTitel) <=
                      FileDateTime(oPath.Backup & "/" & k+1 & "_" & sTitel) THEN
024                   IF k = 1 THEN
025                      i = k
026                      EXIT FOR
027                   END IF
028                ELSE
029                   i = k + 1
030                   EXIT FOR
031                END IF
032             NEXT
033          END IF
034          EXIT FOR
035       END IF
036    NEXT
037    sUrl_Ziel = oPath.Backup & "/" & i & "_" & sTitel
038    FileCopy(sUrl_Start,sUrl_Ziel)
039 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:

```
001 SUB Daten_aus_Cache_schreiben
002    DIM oDaten AS OBJECT
003    DIM oDataSource AS OBJECT
004    oDaten = ThisDatabaseDocument.CurrentController
005    IF NOT ( oDaten.isConnected() ) THEN oDaten.connect()
006    oDataSource = oDaten.DataSource
007    oDataSource.flush
008 END SUB
```

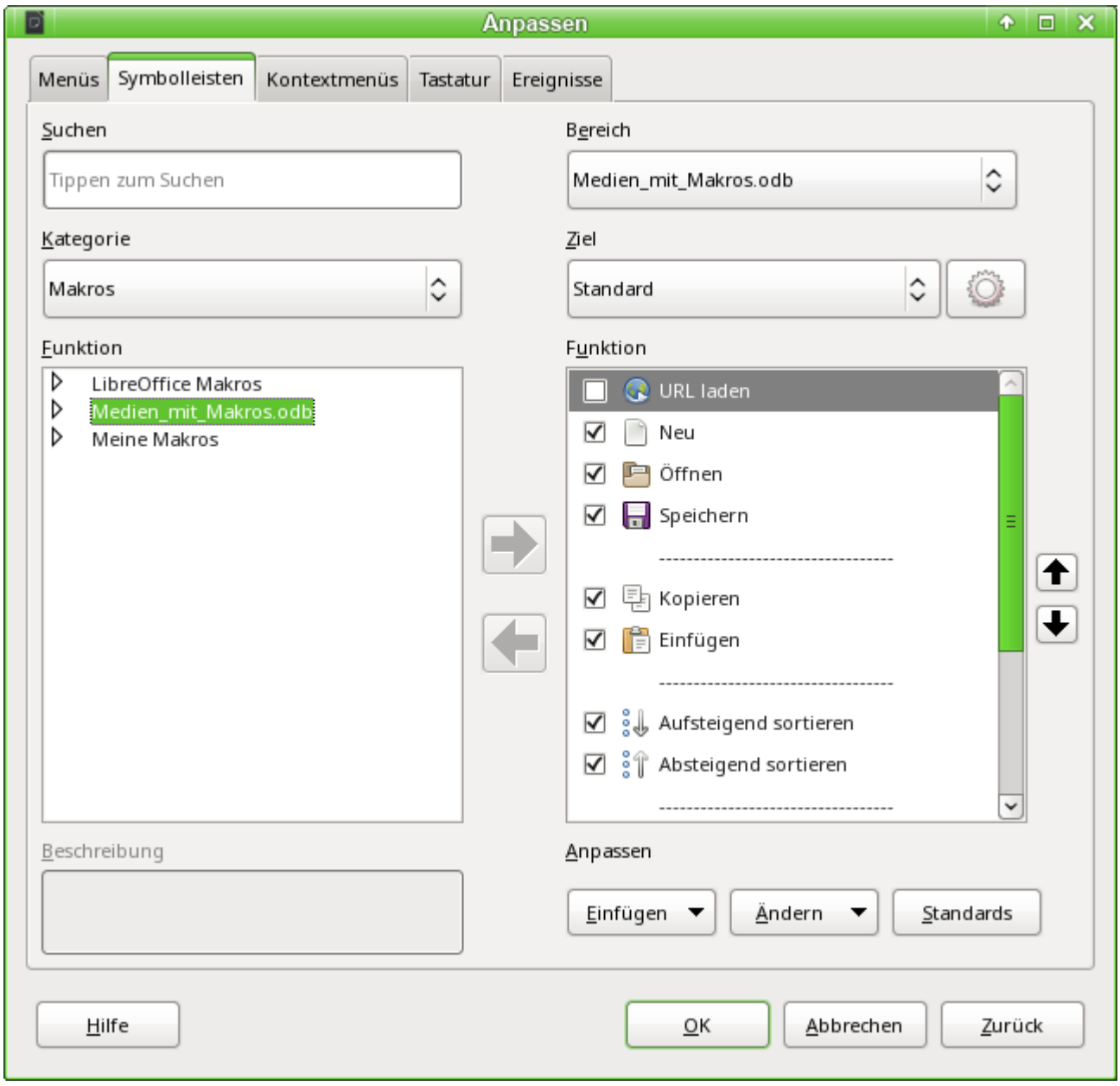
Hinweis

Für Firebird lässt sich dieser Code gut nutzen um beim Schließen der Datenbankdatei die Daten automatisch zu schreiben. Sonst erscheint nach Datenänderungen nämlich die Aufforderung zum Speichern, da das im Gegensatz zu internen Hsqldb nicht automatisch abläuft.
Die Prozedur «Daten_aus_Cache_schreiben» wird unter **Extras → Anpassen → Ereignisse → Ansicht wird geschlossen** eingebunden. Dann erfolgt die Datenspeicherung vor der Abfrage zur Speicherung der Datei.

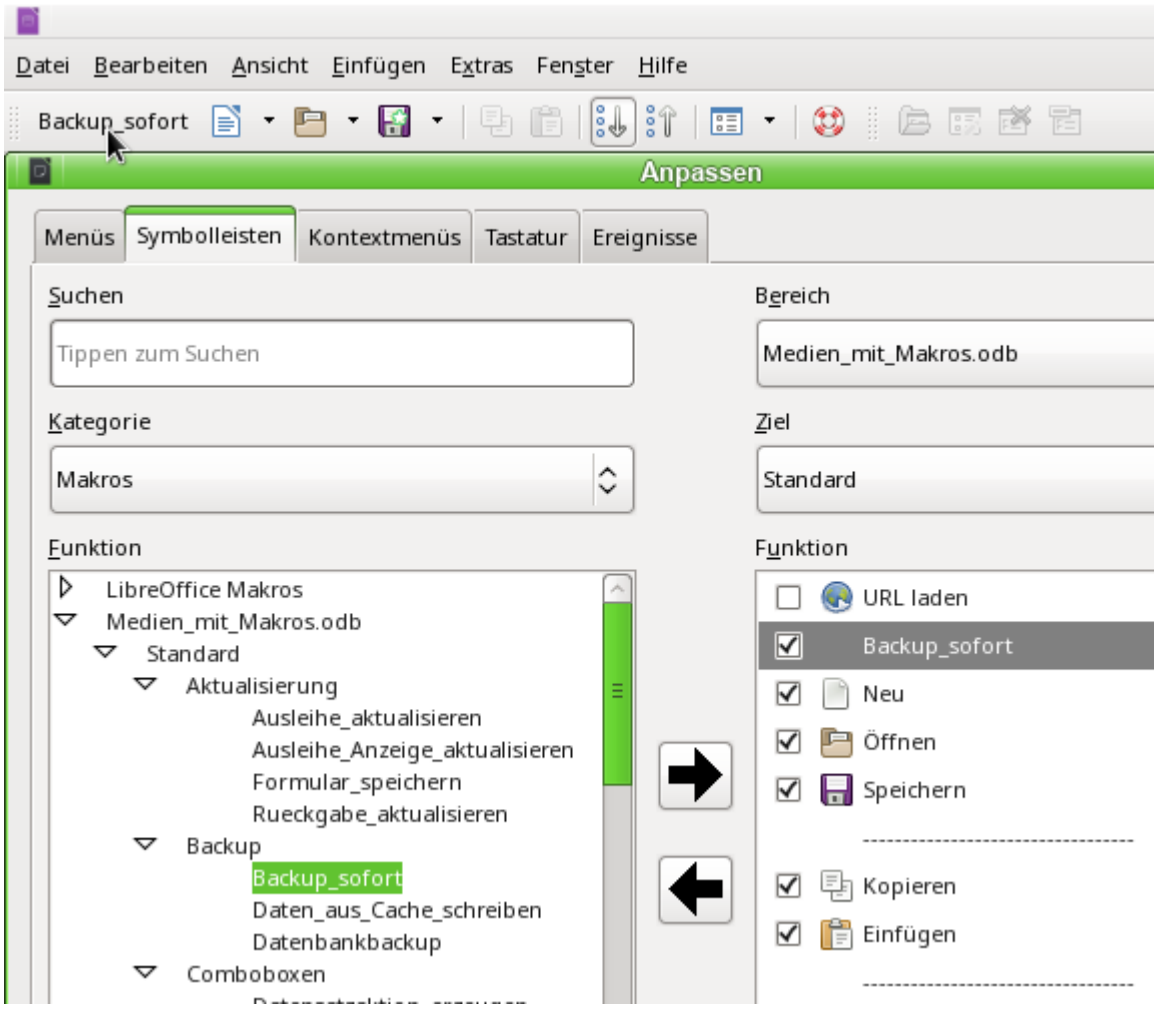
Soll alles zusammen aus einem Formular heraus über einen Button gestartet werden, so müssen beide Prozeduren über eine weitere Prozedur angesprochen werden:

```
001 SUB Backup_sofort
002    Daten_aus_Cache_schreiben
003    Datenbankbackup(10)
004 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.



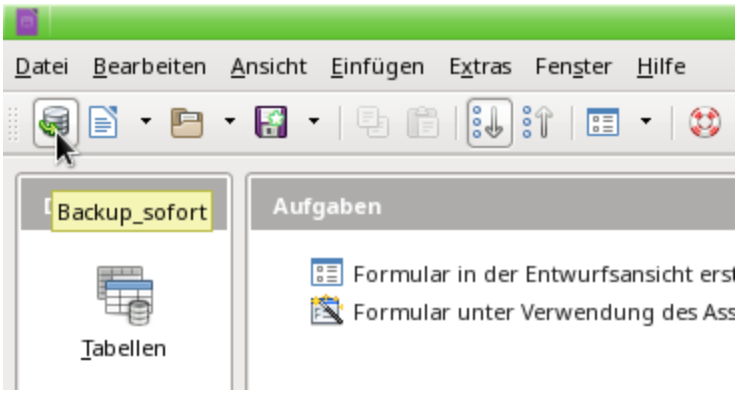
Bei den Makros sind die verfügbaren allgemeinen Makros sowie die Makros aus der Datenbankdatei auswählbar. Aus den Datenbankmakros wird die Prozedur «Backup_Sofort» ausgesucht.



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.

Interne Datenbanken sicher schließen

Die internen Datenbanken müssen vor dem Schließen erst einmal den Inhalt aus dem Speicher zurück in den Datenbankcontainer, die *.odb-Datei, schreiben. Ein einfaches **oDoc.close(True)** führt hier bei dem Datenbankdokument dazu, dass es zwar geschlossen wird, die Änderungen seit dem letzten Speichern nicht mehr gesichert sind. Hierzu muss über **oDoc.Datasource.flush** der Speicherinhalt in die interne Datenbank geschrieben und dann die Speicherung vollzogen werden. Die obige Prozedur Daten_aus_Cache_schreiben muss also für das Schließen zum Schluss nach dem Schreiben aus dem Cache nur um den Eintrag **ThisDatabaseDocument.close(True)** erweitert werden.

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.

```
001  SUB Tabellenindex_runter(stTabelle AS STRING)
002     REM Mit dieser Prozedur wird das automatisch hochgeschriebene
           Primärschlüsselfeld mit der vorgegebenen Bezeichnung "ID" auf den niedrigst
           möglichen Wert eingestellt.
003     DIM stAnzahl AS STRING
004     DIM inAnzahl AS INTEGER
005     DIM inSequence_Value AS INTEGER
006     oDatenquelle = ThisComponent.Parent.CurrentController
           ' Zugriffsmöglichkeit aus dem Formular heraus
007     IF NOT (oDatenquelle.isConnected()) THEN
008        oDatenquelle.connect()
009     END IF
010     oVerbindung = oDatenquelle.ActiveConnection()
011     oSQL_Anweisung = oVerbindung.createStatement()
012     stSql = "SELECT MAX(""ID"") FROM """+stTabelle+""""
           ' Der höchste in "ID" eingetragene Wert wird ermittelt
013     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)    ' Abfrage starten und
           den Rückgabewert in einer Variablen oAbfrageergebnis speichern
014     WHILE oAbfrageergebnis.next
015        stAnzahl = oAbfrageergebnis.getString(1) ' Erstes Datenfeld wird ausgelesen
```

```
016     WEND   ' nächster Datensatz, in diesem Fall nicht mehr erforderlich, da nur ein
           Datensatz existiert
017     IF stAnzahl = "" THEN   ' Falls der höchste Wert gar kein Wert ist, also die
           Tabelle leer ist wird der höchste Wert als -1 angenommen
018        inAnzahl = -1
019     ELSE
020        inAnzahl = cInt(stAnzahl)
021     END IF
022     inSequence_Value = inAnzahl+1  ' Der höchste Wert wird um 1 erhöht
023     REM Ein neuer Befehl an die Datenbank wird vorbereitet. Die ID wird als neu
           startend ab inAnzahl+1 deklariert.
024     REM Diese Anweisung hat keinen Rückgabewert, da ja kein Datensatz ausgelesen
           werden muss
025     oSQL_Anweisung1 = oVerbindung.createStatement()
026     oSQL_Anweisung1.executeQuery("ALTER TABLE """ + stTabelle + """
027        ALTER COLUMN ""ID"" RESTART WITH " + inSequence_Value + "")
028  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:

```
001  SUB Berichtsstart
002     ThisDatabaseDocument.ReportDocuments.getByName("Bericht").open
003  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 gefiltert 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.

```
001  SUB BerichtStart(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM stSql AS STRING
004     DIM oDatenquelle AS OBJECT
005     DIM oVerbindung AS OBJECT
006     DIM oSQL_Anweisung AS OBJECT
007     DIM oReport AS OBJECT
008     DIM oReportView AS OBJECT
009     oForm = oEvent.Source.model.parent
010     stSql = "UPDATE ""Filter"" SET ""Integer"" = '" +
           oForm.getInt(oForm.findColumn("ID")) + "' WHERE ""ID"" = TRUE"
```

```
011     oDatenquelle = ThisComponent.Parent.CurrentController
012     If NOT (oDatenquelle.isConnected()) THEN
013        oDatenquelle.connect()
014     END IF
015     oVerbindung = oDatenquelle.ActiveConnection()
016     oSQL_Anweisung = oVerbindung.createStatement()
017     oSQL_Anweisung.executeUpdate(stSql)
018     oReport = ThisDatabaseDocument.ReportDocuments.getByName("Berichtsname").open
019     oReportView = oReport.CurrentController.Frame.ContainerWindow
020     oReportView.Visible = False
021     BerichtZeilenhoeheAuto(oReport)
022  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.

Hinweis

Diese Prozedur ist ab LO 6.4 nicht mehr notwendig. Seitdem ist im Report-Designer für aufgezogene Felder eine automatische Größeneinstellung möglich. Aber Achtung: Das funktioniert nur ab LO 6.4 und neuer. Die Höhe wird bei älteren Versionen nicht automatisch eingestellt.

Die Zeilenhöhe kann beim Berichtsentwurf bis LO 6.4 nicht 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.

```
001  SUB BerichtZeilenhoeheAuto(oReport AS OBJECT)
002     DIM oTables AS OBJECT
003     DIM oTable AS OBJECT
004     DIM inT AS INTEGER
005     DIM inI AS INTEGER
006     DIM oRows AS OBJECT
007     DIM oRow AS OBJECT
008     oTables = oReport.getTextTables()
009     FOR inT = 0 TO oTables.count() - 1
010        oTable = oTables.getByIndex(inT)
011        IF Left$(oTable.name, 6) = "Detail" THEN
012           oRows = oTable.Rows
013           FOR inI = 0 TO oRows.count - 1
014              oRow = oRows.getByIndex(inI)
015              oRow.IsAutoHeight = True
016           NEXT inI
017        ENDIF
018     NEXT inT
```

```
019     BerichtDruckenUndSchliessen(oReport)
020 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.

```
001 Sub BerichtDruckenUndSchliessen(oReport AS OBJECT)
002     DIM Props
003     DIM stDrucker AS STRING
004     Props = oReport.getPrinter()
005     stDrucker = Props(0).value
006     DIM arg(1) AS NEW com.sun.star.beans.PropertyValue
007     arg(0).name = "Name"
008     arg(0).value = "<" & stDrucker & ">"
009     arg(1).name = "Wait"
010     arg(1).value = True
011     oReport.print(arg())
012     oReport.close(true)
013 End Sub
```

Erst wenn der Druck komplett an den Drucker abgeschickt wurde, wird das Dokument geschlossen.

Zu Einstellungen des Druckers siehe *Drucker und Druckeinstellungen* aus der LibreOffice API.

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:

```
001 Sub BerichtAlsPDFspeichern(oReport AS OBJECT)
002     DIM stUrl AS STRING
003     DIM arg(0) AS NEW com.sun.star.beans.PropertyValue
004     arg(0).name = "FilterName"
005     arg(0).value = "writer_pdf_Export"
006     stUrl = ConvertToURL("file:///....")
007     oReport.storeToURL(stUrl, arg())
008 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.

```
001 SUB Berichtsstart(oEvent AS OBJECT)
002     DIM oFeld AS OBJECT
003     DIM oForm AS OBJECT
004     DIM oDocument AS OBJECT
005     DIM oDocView AS OBJECT
006     DIM Arg()
007     oFeld = oEvent.Source.Model
008     oForm = oFeld.Parent
009     sURL = oForm.DataSourceName
```

```
010     oDocument = StarDesktop.loadComponentFromURL(sURL, "_blank", 0, Arg() )
011     oDocView = oDocument.CurrentController.Frame.ContainerWindow
012     oDocView.Visible = False
013     oDocument.getCurrentController().connect
014     Wait(100)
015     oDocument.ReportDocuments.getByName("Bericht").open
016     oDocument.close(True)
017 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-Designer 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.

```
001 SUB Serienbriefdruck
002     DIM oMailMerge AS OBJECT
003     DIM aProps()
004     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"

```
005     oMailMerge.DataSourceName = "Adressen"
```

Die Pfadbeschreibung mit der Serienbriefdatei erfolgt in der Art der jeweiligen Betriebssystemumgebung, hier ab dem Wurzelpfad eines Linux-Systems.

```
006     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.

```
007     oMailMerge.CommandType = 1
```

Hier wurde eine Abfrage gewählt, die den Namen "Serienbriefabfrage" trägt.

```
008     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 wei-

tergegeben 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.

```
009     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.

```
010     oMailMerge.OutputType = 2
011     oMailMerge.OutputUrl = ConvertToUrl("home/user/Dokumente")
012     oMailMerge.FileNameFromColumn = True
013     oMailMerge.Filenameprefix = "Nachname"
014     oMailMerge.execute(aProps())
015 END SUB
```

Wird allein der Filter über ein Formular bestückt, so kann auf diese Art und Weise, also ohne die Öffnung des Writer-Dokuments, ein Serienbriefdruck erfolgen. Mit der entsprechenden Eingabe von anderen Parametern ist es auch möglich, direkt eine Mail mit persönlichem Anhang zu erstellen. Siehe zu den Parametern *https://api.libreoffice.org/docs/idl/ref/servicecom_1_1sun_1_1star_1_1text_1_1MailMerge.html*.

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-Designer 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*

```
001 SUB Textfelder_Fuellen
002     oForm = thisComponent.Drawpage.Forms.MainForm
003     IF oForm.RowCount = 0 THEN
004        msgbox "Kein Datensatz zum Drucken vorhanden"
005        EXIT SUB
006     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.

```
007     oColumns = oForm.Columns
008     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.

```
009     stDir = Left(oDB.Location,Len(oDB.Location)-Len(ConvertToURL(oDB.Title))+8)
```

Der Titel der Datenbank wird von der URL abgetrennt.

```
010     stDir = stDir & "Beispiel_Textfelder.ott"
```

Die Vorlage wird aufgesucht und geöffnet

```
011     DIM args(0) AS NEW com.sun.star.beans.PropertyValue
012     args(0).Name = "AsTemplate"
013     args(0).Value = True
014     oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
```

Die Textfelder werden eingelesen

```
015     oTextfields = oNewDoc.Textfields.createEnumeration
016     DO WHILE oTextfields.hasMoreElements
017        oTextfield = oTextfields.nextElement
018        IF oTextfield.supportsService("com.sun.star.text.TextField.JumpEdit") THEN
019           stColumnname = oTextfield.PlaceHolder
```

Placeholder ist die Benennung für das Textfeld

```
020           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.

```
021              inIndex = oForm.findColumn(stColumnname)
022              oTextfield.Anchor.String = oForm.getString(inIndex)
023           END IF
024        END IF
025     LOOP
026  END SUB
```

Drucken über Tabellen in Writer

In Tabellenansichten zu drucken liegt bei den vielen Tabellen in einem Datenbankdokument ja nahe. Eine solche Konstruktion über den Report-Designer zu bewerkstelligen ist allerdings eine echte Geduldsprobe. Schon die Positionierung der Felder zusammen mit den Tabellenköpfen zu einem einheitlichen Bild ist wegen der internen Maße in Punkt statt in Zentimeter ein Problem. Eine Umrandungsfunktion für die Zellen gibt es nicht, eine senkrechte oder waagerechte Linie gibt es nur in der Farbe Schwarz als Haarlinie. Und ist so ein Dokument schließlich erstellt, dann dauert der Bericht deutlich länger als mit einer vorgefertigten Writer-Tabelle, weil ein Bericht aus vielen kleinen Tabellen zusammengesetzt wird. Der Druck über Tabellen im Writer[19] wird im Folgenden aufeinander aufbauend an Beispielen erklärt.

Einfacher Druck von Text in Tabellen

Adressen

ID	Vorname	Nachname	Straße	Nr	PLZ	Ort
1	Erika	Schlank	Winkelstr.	3b	34567	Obersumpfingen
2	Marleen	Water	Am Deich	17	28671	Allessiel
3	Jürgen	Gibtsnich	Zur Alm	7	87654	Waldübertal

Für den Druck wird eine Vorlage erstellt, die lediglich aus der Überschrift und einer leeren Tabelle besteht. Die Tabelle besteht in der Vorlage aus der Titelzeile und einer weiteren Zeile für den Inhalt. Die entsprechende Formatierung wird in der Vorlage vorgenommen. Der Tabelle

19 Die Beispieldatenbank «Beispiel_Druck_Writer_Tabellen.odb» liegt den Beispieldatenbanken bei. Jede Druckform ist dort in einem separaten Modul unter gebracht und kommentiert. Notwendige Vorlagen liegen ebenfalls bei. Selbst die Erstellung eines Drucks ohne Vorlage ist in der Beispieldatenbank enthalten, hier aber nicht weiter erklärt. Damit können dann nacheinander mehrere Tabellen erstellt werden. Eine Gruppierung wie im Report-Designer ist möglich.

wird dabei über **Tabelleneigenschaften → Tabelle → Eigenschaften → Name** ein fester Name zugewiesen. Über diesen festen Namen wird die Tabelle im Makro gefunden und mit Inhalt gefüllt.

```
001 SUB FillTable
002    DIM oDB AS OBJECT, oNewDoc AS OBJECT, oTables AS OBJECT, oTable AS OBJECT
003    DIM oRows AS OBJECT, oDatasource AS OBJECT, oConnection AS OBJECT
004    DIM oSQL_Statement AS OBJECT, oResult AS OBJECT
005    DIM stDir AS STRING, stSql AS STRING, stText AS STRING
006    DIM i AS INTEGER, k AS INTEGER, inCols AS INTEGER
```

Der Zugriff auf die Datenbankdatei erfolgt vom Formular aus. Die *.odb-Datei ist Parent des Formulardokumentes. Anschließend wird daraus der Pfad ermittelt, in dem die Vorlagendatei liegt. In diesem Beispiel liegt die Vorlage im gleichen Pfad wie Datenbankdatei.

```
007    oDB = ThisComponent.Parent
008    stDir = Left(oDB.Location,Len(oDB.Location)-Len(ConvertToURL(oDB.Title))+8)
```

Der Titel der Datenbank wird von der URL abgetrennt. Die Längenermittlung mit **ConvertToURL** ist notwendig, falls der Titel Leerzeichen enthält.

```
009    stDir = stDir & "PrintStart.ott"
```

Die Vorlage wird geöffnet und das Writer-Dokument damit erstellt.

```
010    DIM args(0) AS NEW com.sun.star.beans.PropertyValue
011    args(0).Name = "AsTemplate"
012    args(0).Value = True
013    oNewDoc = StarDesktop.loadComponentFromURL(stDir,"_blank",0,args)
```

Die Vorlage enthält eine Tabelle, die den Namen «Printout_Addresses» hat. Diese Tabelle besteht aus 2 Zeilen - der Titelzeile und der ersten Zeile für den Inhalt

```
014    oTables = oNewDoc.getTextTables
015    oTable = oTables.getByName("Printout_Addresses")
016    inCols = oTable.Columns.Count
```

Mit der Variablen **i** wird die Inhaltszeile angesteuert. Die Zeilenzählung bei Tabellen beginnt mit der Zeile 0.

```
017    i = 1
018    oDatasource = ThisComponent.Parent.CurrentController
019    If NOT (oDatasource.isConnected()) THEN
020       oDatasource.connect()
021    END IF
022    oConnection = oDatasource.ActiveConnection()
023    oSQL_Statement = oConnection.createStatement()
024    stSql = "SELECT * FROM ""tbl_Adressen"""
025    oResult = oSQL_Statement.executeQuery(stSql)
```

Die Daten aus "tbl_Adressen" werden ausgelesen. Anschließend werden die Inhalte direkt in die entsprechenden Zellen der Tabelle als Text eingefügt.

```
026    WHILE oResult.next
027       FOR k = 0 TO inCols-1
028          stText = oResult.getString(k+1)
029          oTable.getCellByPosition(k,i).setString(stText)
030       NEXT
```

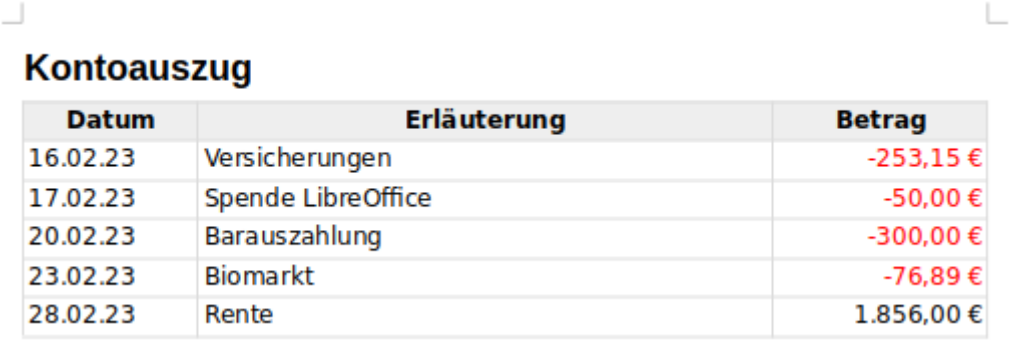
Für jeden neuen Datensatz muss eine zusätzliche Inhaltszeile erstellt werden.

```
031       oRows = oTable.getRows()
032       oRows.insertByIndex(oRows.getCount(),1)
033       i = i + 1
034    WEND
```

Durch die Schleife bleibt zum Schluss eine leere Inhaltszeile übrig. Diese Zeile könnte weiter genutzt werden (z.B. für eine Summierung bei Rechnungen). Hier ist die Zeile nicht nötig und wird deswegen entfernt.

```
035    oRows.removeByIndex(oRows.getCount()-1,1)
036 END SUB
```

Tabellendruck mit formatierten Zellen



Kontoauszug

Datum	Erläuterung	Betrag
16.02.23	Versicherungen	-253,15 €
17.02.23	Spende LibreOffice	-50,00 €
20.02.23	Barauszahlung	-300,00 €
23.02.23	Biomarkt	-76,89 €
28.02.23	Rente	1.856,00 €

Bereits mit einer kleinen Erweiterung lässt sich das entsprechende Format für die Spalten erstellen. Hierzu wird in der Vorlage die Spalte für das Datum (Zelle A2) als Datum formatiert, die Spalte für den Betrag (Zelle C2) als Währungsfeld formatiert. Außerdem wird noch die Spalte für den Betrag rechtsbündig gesetzt.

Alle weiteren durch das Makro hinzugefügten Zeilen übernehmen die Formatierung aus der vorhergehenden Zeile. Der Makro-Code wird um einige Zeilen ergänzt.

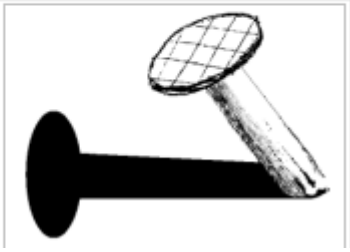
```
024     stSql = "SELECT ""Datum"", ""Erläuterung"", ""Betrag"" FROM ""tbl_Konto"""
025     arFormat = array("Date","String","Number")
026     oResult = oSQL_Statement.executeQuery(stSql)
```

Die Daten aus "tbl_Konto" werden als Text ausgelesen. Anschließend werden die Inhalte in Abhängigkeit vom gewünschten Format (siehe das Array in Zeile 25) als Wert oder Text eingefügt. Die Werte für das Datum und die Zahl werden über die Funktionen **CDate** und **Val** berechnet. Das Einfügen der Werte erfolgt im Gegensatz zum Text mit **setValue**.

```
027     WHILE oResult.next
028        FOR k = 0 TO inCols-1
029           stText = oResult.getString(k+1)
030           SELECT CASE arFormat(k)
031              CASE "Date"
032                 oTable.getCellByPosition(k,i).setValue(CDate(stText))
033              CASE "Number"
034                 oTable.getCellByPosition(k,i).setValue(Val(stText))
035              CASE "String"
036                 oTable.getCellByPosition(k,i).setString(stText)
037           END SELECT
038        NEXT
039        oRows = oTable.getRows()
040        oRows.insertByIndex(oRows.getCount(),1)
041        i = i + 1
042     WEND
```

Tabellendruck mit Bildern in der Tabelle

Tabelle mit Bildern

ID	Name	Inhalt	Bild
1	Stern	Gezeichnet in Draw, exportiert als *.png-Datei	
2	Nägel	Eine Datei im tif-Format	

Das Einfügen von Bildern aus der Datenbank in die Tabelle ist etwas aufwändiger. Sind die Bilder in die Tabelle der Datenbank eingelesen, so müssen sie erst einmal als externe Bilder ausgelesen werden. Danach erfolgt dann das Einfügen des Bildes in die Tabelle. Damit das Bild auch so wie im Screenshot positioniert wird muss zum Schluss noch die Verankerung von der Verankerung **Am Absatz** zur Verankerung **Als Zeichen** geändert werden.

```
028         FOR k = 0 TO inCols-1
029            SELECT CASE arFormat(k)
030               CASE "Date"
031                  stText = oResult.getString(k+1)
032                  oTable.getCellByPosition(k,i).setValue(CDate(stText))
033               CASE "Number"
034                  stText = oResult.getString(k+1)
035                  oTable.getCellByPosition(k,i).setValue(Val(stText))
036               CASE "String"
037                  stText = oResult.getString(k+1)
038                  oTable.getCellByPosition(k,i).setString(stText)
039               CASE "Image"
040                  stText = oResult.getString(k+1)
041                  ImportImage(stText,oNewDoc,oTable.getCellByPosition(k,i))
042               CASE "ImageIntern"
043                  oStream = oResult.getBinaryStream(k+1)
044                  ImportImageIntern(oStream,oNewDoc,oTable.getCellByPosition(k,i))
045            END SELECT
046         NEXT
```

in der Hauptprozedur werden lediglich zwei zusätzliche Formattypen eingefügt.

Ist in der Datenbank lediglich der Link zu dem Bild abgespeichert, dann wird aus der Abfrage ein Text ausgelesen und dieser Link zusammen mit dem Objekt des Writer-Dokumentes und der genauen Position an die Prozedur **`ImportImage`** weiter gereicht.

Handelt es sich dagegen um ein Bild, das in die Datenbank eingelesen wurde, so muss der Inhalt über **`getBinaryStream`** ausgelesen werden. Dieser Stream wird genutzt, um ihn als Bilddatei im temporären Pfad von LibreOffice ab zu speichern.

```
001 SUB ImportImageIntern(oStream,oNewDoc,oPos)
002    DIM oPath AS OBJECT, oSimpleFileAccess AS OBJECT
003    DIM stPath AS STRING
```

Das ausgelesene Bild wird im temporären Pfad unter dem Namen **DbFile** zwischengespeichert. Anschließend wird die Prozedur aufgerufen, die das Bild in das Writer-Dokument übernimmt.

```
004    oPath = createUnoService("com.sun.star.util.PathSettings")
005    stPath = oPath.Temp & "/DbFile"
006    oSimpleFileAccess = createUnoService("com.sun.star.ucb.SimpleFileAccess")
007    oSimpleFileAccess.writeFile(stPath, oStream)
008    ImportImage(stPath,oNewDoc,oPos)
009 END SUB
```

Die Prozedur zum Einfügen des Bildes in die Tabelle benötigt die Informationen über den Pfad zum Bild, das Writer-Dokument und die genaue Position innerhalb der Tabelle.

```
001 SUB ImportImage(stPath,oNewDoc,oPos)
002    DIM oCursor AS OBJECT, oTextRange AS OBJECT, oDocCtrl AS OBJECT
003    DIM oDispatcher AS OBJECT, oGraphic AS OBJECT
```

Der sichtbare Cursor wird in die Tabellenzelle gesetzt. Zuerst wird die Position am Ende des Absatzes gewählt und dann der Cursor positioniert.

```
004    oCursor = oNewDoc.CurrentController.ViewCursor
005    oTxtRange = oPos.getEnd
006    oCursor.goToRange(oTxtRange,False)
```

Um das Bild einzufügen muss auf den den Controller zugegriffen werden. Anschließend wird über den **Dispatcher** mit **uno:InsertGraphic** das Bild eingelesen. Die wichtigsten Eigenschaften sind hier benannt: Der Pfad zum Bild und dass das Bild nicht als Verknüpfung eingefügt wird. Prinzipiell ist auch das Einlesen von Bildern über die **Drawpage** der Seite möglich. Nur nimmt der **Dispatcher** hier sehr viel Arbeit ab: Das Bild wird korrekt positioniert, die Größe des Bildes wird maximal der Spaltenbreite angepasst und die Größenverhältnisse werden automatisch beibehalten.

```
007    oDocCtrl   = oNewDoc.CurrentController.Frame
008    oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
009    DIM args(1) AS NEW com.sun.star.beans.PropertyValue
010    args(0).Name = "FileName"
011    args(0).Value = stPath
012    args(1).Name = "AsLink"
013    args(1).Value = false
014    oDispatcher.executeDispatch(oDocCtrl, ".uno:InsertGraphic", "", 0, args())
```

Auf die gerade eingefügte Grafik wird über den Index zugegriffen, damit der Anker gesetzt werden kann. Die Standardverankerung an den Absatz erzeugt sonst eine leere Zeile unterhalb des Bildes, so dass der Abstand zur Tabellenzeile nach unten zu groß wird.

```
015    oGraphic =
          oNewDoc.getGraphicObjects.getByIndex(oNewDoc.getGraphicObjects.Count - 1)
```

Mit **AnchorType = 1** wird die Verankerung als Zeichen gesetzt. Der Cursor wird direkt nach dem Bild positioniert und erzeugt keinen zusätzlichen Rand.

```
016    oGraphic.AnchorType = 1
017 END SUB
```

Hier die verschiedenen möglichen Ankertypen als Zusatzinformation:

- 0 → AT_PARAGRAPH
- 1 → AS_CHARACTER
- 2 → AT_PAGE
- 3 → AT_FRAME
- 4 → AT_CHARACTER

Siehe: *https://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1text.html#a470b1caeda4ff15fee438c8ff9e3d834*

Rechnungen mit Übertrag im Tabellendruck

	LI-Ionen-Akkus und Akkuladegerät		
1	Spiegelreflex-Kamera mit Zoomoptik 25-70mm incl. Speicherkarte 32GB, LI-Ionen-Akkus und Akkuladegerät	834,56 €	834,56 €
		Übertrag:	*9.640,50 €*
		Übertrag:	*9.640,50 €*
1	Spiegelreflex-Kamera	834,56 €	834,56 €

Eine Rechnung mit Übertrag auszudrucken ist über den Report-Designer nur möglich, wenn die Zeilen nicht automatisch anwachsen. Dann kann genau vorausberechnet werden, an welcher Stelle der Seitenumbruch erfolgt und wann dann ein Übertrag eingeblendet werden muss. Sobald die Zeilen der Tabelle sich automatisch an den Textinhalt anpassen und damit größer werden können funktioniert ein Ausdruck wie im obigen Screenshot nicht mehr mit dem Report-Designer.

Die ursprüngliche Prozedur für das Einfügen von Inhalt in eine Tabelle wird erweitert. Zuerst wird das Linienformat für die Absetzung von Schlusssumme und eventuell Überträgen (**TopBorder**, **BottomBorder**) erstellt.

```
018     oBorder = CreateUnoStruct("com.sun.star.table.BorderLine2")
019     oBorder.Color = 0
020     oBorder.LineWidth = 1
021     WHILE oResult.next
022        FOR k = 0 TO inCols-1
023           stText = oResult.getString(k+1)
024           SELECT CASE arFormat(k)
025              CASE "Number"
026                 doValue = Val(stText)
027                 oTable.getCellByPosition(k,i).setValue(doValue)
```

In der 4. Spalte soll die Aufsummierung erfolgen:

```
028                 IF k = 3 THEN
```

Die Summe des vorhergehenden Durchgangs wird in **doCarryOver** gespeichert. Der gerade neu ausgelesene Wert wird zu der vorhergehenden Summe addiert und in **doVal** gespeichert.

```
029                    doCarryOver = doVal
030                    doVal = doVal + doValue
031                 END IF
032              CASE "String"
033                 oTable.getCellByPosition(k,i).setString(stText)
034           END SELECT
035        NEXT
036        oRows = oTable.getRows()
037        oRows.insertByIndex(oRows.getCount(),1)
038        i = i + 1
```

Nachdem eine neue Tabellenzeile eingefügt wurde muss der sichtbare Cursor in diese Zeile gesetzt werden. Für die neue Zeile wird die Seitenzahl bestimmt. Ist die Seitenzahl größer als die vorhergehende Seitenzahl, so muss ein Übertrag eingefügt werden. Der Übertrag muss vor der vorhergehenden Zeile erfolgen.

```
039         oPos = oTable.getCellByPosition(0,i)
040         oTxtRange = oPos.getEnd
041         oCursor.goToRange(oTxtRange,False)
042         IF oCursor.Page > inPageStart THEN
```

Es werden 2 leere Zeile oberhalb des letzten Eintrags eingefügt. Beide Zeilen erhalten den Eintrag "Übertrag:" in der 3. Spalte und den Wert für den Übertrag in der 4. Spalte. Für die entsprechenden Zellen wird das Absatzformat "Übertrag" gesetzt. Achtung: Die Schriftart für das Absatzformat "Übertrag" darf für die Positionierung nicht größer sein als die für den Tabelleninhalt.

```
043             FOR ink = 1 TO 2
044                oRows = oTable.getRows()
045                oRows.insertByIndex(oRows.getCount()-2,1)
046                oTable.getCellByPosition(2,i-1).setString("Übertrag:")
047                oTable.getCellByPosition(3,i-1).setValue(doCarryOver)
048                oTable.getCellByPosition(2,i-1).getEnd.ParaStyleName = "Übertrag"
049                oTable.getCellByPosition(3,i-1).getEnd.ParaStyleName = "Übertrag"
050                i = i + 1
051             NEXT
```

Der Cursor wird jetzt von der unteren leeren Tabellenzeile 2 Zeilen nach oben auf die 2. Zeile für den Übertrag gesetzt. Die 2. Zeile für den Übertrag darf nicht auf der gleichen Seite wie die 1. Zeile für den Übertrag stehen. Die 2. Zeile kann zufällig auf der gleichen Seite wie die erste Zeile stehen, wenn der letzte inhaltliche Eintrag einen Absatzumbruch enthält. In dem Fall nimmt die Tabellenzeile für den letzten Eintrag eine größere Höhe in Anspruch. Die 1. Zeile für den Übertrag steht auf **inPageStart**.

```
052             oCursor.goUp(2,False)
053             DO WHILE oCursor.Page = inPageStart
```

Es werden so lange zwischen der 1. Zeile für den Übertrag und der 2. Zeile für den Übertrag Tabellenzeilen eingefügt, bis die 2. Zeile auf der Folgezeile steht.

```
054                oRows = oTable.getRows()
055                oRows.insertByIndex(oRows.getCount()-3,1)
056                oPos = oTable.getCellByPosition(0,i-1)
057                oTxtRange = oPos.getEnd
058                oCursor.gotoRange(oTxtRange,false)
059                i = i + 1
060             LOOP
```

Die jetzt letzte Seite wird als **inStartPage** zur neuen Vergleichsseite für den nächsten Übertrag.

```
061             inPageStart = oCursor.Page
062          END IF
063       WEND
```

Durch die Schleife bleibt eine leere Inhaltszeile übrig. Diese Zeile wird für die Summierung genutzt. Wie beim Übertrag erfolgen hier die Einträge und die Formatierungen. Zusätzlich wird die oben stehende Umrandung als Abgrenzung für den Gesamtpreis in die Zellen eingefügt.

```
064       oTable.getCellByPosition(2,i).setString("Gesamt:")
065       oTable.getCellByPosition(3,i).setValue(doVal)
066       oTable.getCellByPosition(2,i).getEnd.ParaStyleName = "Übertrag"
067       oTable.getCellByPosition(3,i).getEnd.ParaStyleName = "Übertrag"
068       oTable.getCellByPosition(2,i).TopBorder = oBorder
069       oTable.getCellByPosition(3,i).TopBorder = oBorder
070    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»[20]

```
001 SUB Link_oeffnen
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeld AS OBJECT
006    DIM oShell AS OBJECT
007    DIM stFeld AS STRING
008    oDoc = thisComponent
009    oDrawpage = oDoc.Drawpage
010    oForm = oDrawpage.Forms.getByName("Formular")
011    oFeld = oForm.getByName("Adresse")
```

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 ...).

Hinweis

Der Verwendung von Sonderzeichen in den Pfaden und Dateinamen sollte hier möglichst vermieden werden. Die Links sind dann teilweise auf diese Art nicht mehr aufrufbar.

```
012    stFeld = oFeld.Text
013    IF stFeld = "" THEN
014       EXIT SUB
015    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.

```
016    IF InStr(stFeld,"@") THEN
017       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:///'**

```
018    ELSEIF InStr(stFeld,"http") = 0 THEN
019       stFeld = convertToUrl(stFeld)
020    ELSE
021    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.

```
022    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
023    oShell.execute(stFeld,,0)
024 END SUB
```

20 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 ...

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='**.

```
001 SUB Mail_Aufruf
002    DIM oDoc AS OBJECT
003    DIM oDrawpage AS OBJECT
004    DIM oForm AS OBJECT
005    DIM oFeld1 AS OBJECT
006    DIM oFeld2 AS OBJECT
007    DIM oFeld3 AS OBJECT
008    DIM oFeld4 AS OBJECT
009    DIM oShell AS OBJECT
010    DIM stFeld1 AS STRING
011    DIM stFeld2 AS STRING
012    DIM stFeld3 AS STRING
013    DIM stFeld4 AS STRING
014    oDoc = thisComponent
015    oDrawpage = oDoc.Drawpage
016    oForm = oDrawpage.Forms.getByName("Formular")
017    oFeld1 = oForm.getByName("E_Mail_Adresse")
018    oFeld2 = oForm.getByName("E_Mail_Betreff")
019    oFeld3 = oForm.getByName("E_Mail_Inhalt")
020    stFeld1 = oFeld1.Text
021    IF stFeld1 = "" THEN
022       msgbox "Keine Mailadresse vorhanden." & CHR(13) &
          "Das Mailprogramm wird nicht aufgerufen" , 48, "Mail senden"
023       EXIT SUB
024    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. Die Umwandlung ist bei der Verwendung von **SimpleSystemMail/SimpleCommandMail** (siehe den folgenden Hinweis) nicht erforderlich.

```
025    stFeld2 = Mid(ConvertToUrl(oFeld2.Text),9)
026    stFeld3 = Mid(ConvertToUrl(oFeld3.Text),9)
```

Im Gegensatz zum einfachen Programmaufruf werden hier Details des Mailaufrufes über den Aufruf des Mailprogramms mitgegeben.

```
027    oShell = createUnoService("com.sun.star.system.SystemShellExecute")
028    oShell.execute("mailto:" + stFeld1 + "?subject=" + stFeld2 + "&body=" +
          stFeld3,,0)
029 END SUB
```

Hinweis

Das Versenden von Mails mit Hilfe des Mailprogramms kann auch mit folgendem Code erfolgen. Ab LO 4.2 kann hier auch über das Attribut Body der Inhalt der Mail mit eingefügt werden. Dieser Code ermöglicht außerdem das Anfügen von Anhängen. Für mehrere Anhänge muss einfach das Array erweitert werden. Auch Adressen im CC sowie im BCC werden in ein Array geschrieben.

```
027     DIM attachs(0)
028     IF GetGuiType() = 1 THEN
029        oMailer =
              createUnoService("com.sun.star.system.SimpleSystemMail")
              ' Sonst Linux/Mac
030        ELSE
031        oMailer =
              createUnoService("com.sun.star.system.SimpleCommandMail")
032     END IF
033     oMailProgramm = oMailer.querySimpleMailClient()
034     oNeueNachricht = oMailProgramm.createSimpleMailMessage()
035     oNeueNachricht.setRecipient(stFeld1)
036     oNeueNachricht.setSubject(stFeld2)
037     oNeueNachricht.Body = stFeld3
038     attachs(0) = "file:///..."
039     oNeueNachricht.setAttachement(attachs())
040     oMailprogramm.sendSimpleMailMessage(oNeueNachricht, 0 )
041  END SUB
```

Zu den möglichen Parametern siehe: *http://api.libreoffice.org/docs/idl/ref/interfacecom_1_1sun_1_1star_1_1system_1_1XSimpleMailMessage.html*

Manchmal kommt es unter Linux zu Problemen mit Kommas in **Subject** und **Body**. Der Text wird dadurch einfach abgeschnitten, taucht teilweise auch als Empfänger auf. Auch Zeilenumbrüche können zu Problemen führen.

Auch wenn unter **Extras → Optionen → Internet → E-Mail** das passende Mailprogramm angegeben wurde funktioniert der Versand nicht unbedingt. Unter Ubuntu-Linux 22.04 darf dort kein Eintrag stehen. Es wird immer das Standardmailprogramm geöffnet. Und dies sollte unter Ubuntu nicht Thunderbird (in der Version 115.4.1) sein, da dort dann genau die oben beschriebenen Probleme mit Kommas und Zeilenumbrüchen auftauchen. Mit Evolution gibt es dort keine weiteren Probleme.

Unter OpenSUSE-Linux 15.4 hingegen funktioniert Thunderbird (in der Version 115.4.2) mit dem Makro einwandfrei. Hier ist also ein Testen des eigenen Systems gefragt.

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.[21]

```
001  SUB MapPosition(oEvent AS OBJECT)
002     DIM oForm AS OBJECT, oShell AS OBJECT
003     DIM i AS INTEGER
004     DIM stLink AS STRING, stTag AS STRING
005     DIM arFields()
006     stTag = oEvent.Source.Model.Tag
007     oForm = oEvent.Source.Model.Parent
008     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.

21 Siehe Beispiel_Formular_Eingabekontrolle.odb

```
009     FOR i = LBound(arFields) TO UBound(arFields)
010        IF stLink = "" THEN
011           stLink = oForm.getByName(arFields(i)).CurrentValue
012        ELSE
013           stLink = stLink & "+" & oForm.getByName(arFields(i)).CurrentValue
014        END IF
015     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.

```
016     IF stLink <> "" THEN
017        stLink = "https://nominatim.openstreetmap.org/search.php?q=" &
              Join(Split(stLink),"+") & "&polygon_geojson=1&viewbox="
018        oShell = createUnoService("com.sun.star.system.SystemShellExecute")
019        oShell.execute(stLink,,0)
020     END IF
021  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.[22]

Mauszeiger ändern

Manchmal erscheint es sinnvoll, die Mauszeiger so anzupassen, dass sie Zusatzinformationen zur Verwendung des Inhaltes eines Feldes geben.

Änderung beim Überfahren eines Links

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»[23].

Diese kurze Prozedur sollte mit dem Ereignis **'Maus innerhalb'** des Textfeldes verbunden werden.

```
001  SUB Mauszeiger(oEvent AS OBJECT)
002     REM Siehe auch Standardbibliotheken: Tools → ModuleControls → SwitchMousePointer
003     DIM oPointer AS OBJECT
004     oPointer = createUnoService("com.sun.star.awt.Pointer")
005     oPointer.setType(27) 'Typen in com.sun.star.awt.SystemPointer
006     oEvent.Source.Peer.SetPointer(oPointer)
007  END SUB
```

Änderung bei gedrückter Strg-Taste und Mausklick

```
001  SUB Mauszeiger(oEvent AS OBJECT)
002     DIM oPointer AS OBJECT
003     oPointer = createUnoService("com.sun.star.awt.Pointer")
004     IF oEvent.Modifiers = 2 THEN
005        'KeyModifier (ohne: 0 | Shift: 1 | Ctrl: 2 | Alt: 4 …),
              Typen in com.sun.star.awt.KeyModifier
```

22 Bei Nutzung dieser Möglichkeit der Kartendarstellung sollten die Bedingungen der Website beachtet werden: *https://operations.osmfoundation.org/policies/nominatim/* .

23 Die Datenbank «Beispiel_Mailstart_Dateiaufruf.odb» ist diesem Handbuch beigefügt.

```
006         oPointer.setType(0) 'Typen in com.sun.star.awt.SystemPointer
007      ELSE
008         oPointer.setType(3)
009      END IF
010      oEvent.Source.Peer.SetPointer(oPointer)
011   END SUB
```

Über den **KeyModifier** wird ermittelt, ob eine der entsprechenden Tasten zusätzlich zu dem Mausklick an der Auslösung des Makros beteiligt war. Hier wurde mit '2' als zusätzliche Taste *STRG* ausgewählt. Wird *STRG* nicht gedrückt, so wird auf den Textcursor geschaltet.

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 werden. 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.

```
001   SUB Symbolleisten_Ausblenden
002      DIM oFrame AS OBJECT
003      DIM oWin AS OBJECT
004      DIM oLayoutMng AS OBJECT
005      DIM aElemente()
006      oFrame = StarDesktop.getCurrentFrame()
```

Diese Startvariante ist für **eigenständige Formulare** geeignet, nicht aber für Formulare in der Basedatei. In der Basedatei würde dort das Hauptfenster, nicht aber die dem **untergeordneten Formulare** ohne Symbolleiste versehen. Dort erfolgt der Start über

```
001   SUB Symbolleisten_Ausblenden(oEvent AS OBJECT)
002      DIM oFrame AS OBJECT
003      DIM oWin AS OBJECT
004      DIM oLayoutMng AS OBJECT
005      DIM aElemente()
006      oFrame = oEvent.Source.CurrentController.Frame
```

Der Titel für das Formular wird in der Titelleiste des Fensters angezeigt.

```
007      oFrame.setTitle "Mein Formular"
008      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.

```
009      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.

Da es inzwischen auch viele verschiedene Bildschirmauflösungen gibt und die Fenstergröße eventuell angepasst werden soll, hier die Ermittlung der Bildschirmauflösung in dpi: **'oWin.Info.PixelPerMeterX * 2.54/100'**. Die Auflösung wird genauso für die y-Achse angegeben ist aber vermutlich immer gleich.

```
010     oLayoutMng = oFrame.LayoutManager
011     aElemente = oLayoutMng.getElements()
012     FOR i = LBound(aElemente) TO UBound(aElemente)
013        IF aElemente(i).ResourceURL =
              "private:resource/toolbar/formsnavigationbar" THEN
014        ELSE
015           oLayoutMng.hideElement(aElemente(i).ResourceURL)
016        END IF
017     NEXT
018     ThisComponent.CurrentController.Sidebar.Visible = False
019     ThisComponent.CurrentController.ViewSettings.ZoomValue = 200
020     ThisComponent.CurrentController.ViewSettings.ShowRulers = False
021     ThisComponent.CurrentController.ViewSettings.ShowParaBreaks = False
022  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**. Neben den Symbolleisten wird anschließend noch die Seitenleiste unsichtbar gemacht, in diesem Beispiel dann auch noch der **ZoomValue** eingestellt, die Lineale links und oben ausgeblendet und die Absatzmarke nicht mehr angezeigt, falls sonst im Writer eben Formatierungszeichen angezeigt werden.

Bei unterschiedlichen Bildschirmen kann es passieren, dass der voreingestellte **ZoomValue** nicht die gleiche Formularansicht wiedergibt. Hier kann das Auslesen von Bildschirmbreite und Bildschirmhöhe helfen:

```
018     ' Ausschnitt mit allen Elementen des Formulars: 1487*765
019     ' bei 96 dpi, 3779 PixelPerMeter – gezoomte Bildschirme haben mehr dpi
020     inDpiX = 1440 \ TwipsPerPixelX()
021     inDpiY = 1440 \ TwipsPerPixelY()
022     inx = Int(oWin.Info.Width * 100 * 96 / (1487 * inDpiX))
023     iny = Int(oWin.Info.Height * 100 * 96 / (765 * inDpiY))
024     IF inx < iny THEN
025        inZoom = inx
026     ELSE
027        inZoom = iny
028     END IF
```

Über einen Screenshot wurde die Größe des Formulars (links oben unterhalb der Symbolleisten beginnen bis rechts unten incl. des letzten Elementes; ggf. auch die Navigationsleiste mit einbeziehen) in Pixeln bei einem **ZoomValue** von 100 ermittelt. Das Verhältnis von tatsächlicher Bildschirmbreite zu Formularbreite bzw. Bildschirmhöhe zu Formularhöhe soll den neuen Prozentwert für den **ZoomValue** bestimmen. Damit auch das gesamte Formular auf den Bildschirm passt soll der kleinere Wert übernommen werden. Das Ganze wird in die vorhergehende Prozedur nach der Deklaration von **oWin** eingebaut. Statt der '200' für den **ZoomValue** steht dort dann eben **inZoom**. Alle hier auftauchenden Zahlenvariablen sind Ganzzahlen im Integer-Format.

Bei Addons im Bereich der Symbolleisten wird die Eigenschaft **ResourceURL** leider etwas hinter einer Integer-Variablen versteckt. Hier ist dann zur Bestimmung der URL der folgende Weg notwendig:

```
016     obj = aElemente(i)
017     invoc = CreateUnoService("com.sun.star.script.Invocation")
018     invocCurrObj = invoc.createInstanceWithArguments(Array(obj))
019     ResourceURL= invocCurrObj.getValue("ResourceURL")
020     oLayoutMng.hideElement(ResourceURL)
```

Dieser Code sollte gegebenenfalls in die **FOR**-Schleife vor **END IF** eingefügt werden.

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.

```
001 SUB Symbolleisten_Einblenden
002    DIM oFrame AS OBJECT
003    DIM oLayoutMng AS OBJECT
004    DIM aElemente()
005    oFrame = StarDesktop.getCurrentFrame()
006    oLayoutMng = oFrame.LayoutManager
007    aElemente = oLayoutMng.getElements()
008    FOR i = LBound(aElemente) TO UBound(aElemente)
009       oLayoutMng.showElement(aElemente(i).ResourceURL)
010    NEXT
011    ' eventuell fehlende wichtige Elemente:
012    ' "private:resource/toolbar/standardbar"
013    ' "private:resource/statusbar/statusbar"
014    ThisComponent.CurrentController.Sidebar.Visible = True
015    ThisComponent.CurrentController.ViewSettings.ZoomValue = 100
016    ThisComponent.CurrentController.ViewSettings.ShowRulers = True
017    ThisComponent.CurrentController.ViewSettings.ShowParaBreaks = True
018 END SUB
```

Die Makros werden an die Eigenschaften des Formularfensters gebunden: **Extras → Anpassen → Ereignisse → Dokument öffnen → Symbolleisten_Ausblenden** bzw. **… Dokument wird geschlossen → Symbolleisten_Einblenden**

Auch diese Prozedur sowie die folgende muss in internen Formularen von Base anders gestartet werden, da sonst das Hauptfenster eingestellt wird.

```
001 SUB Symbolleisten_Einblenden(oEvent AS OBJECT)
      ' …
002    oFrame = oEvent.Source.CurrentController.Frame
```

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:

```
001 Sub Symbolleisten_Einblenden
002    DIM oFrame AS OBJECT
003    DIM oLayoutMng AS OBJECT
004    DIM i AS INTEGER
005    DIM aElemente(5) AS STRING
006    oFrame = StarDesktop.getCurrentFrame()
007    oLayoutMng = oFrame.LayoutManager
008    aElemente(0) = "private:resource/menubar/menubar"
009    aElemente(1) = "private:resource/statusbar/statusbar"
010    aElemente(2) = "private:resource/toolbar/formsnavigationbar"
011    aElemente(3) = "private:resource/toolbar/standardbar"
012    aElemente(4) = "private:resource/toolbar/formdesign"
013    aElemente(5) = "private:resource/toolbar/formcontrols"
014    FOR i = LBound(aElemente) TO UBound(aElemente)
015       IF NOT(oLayoutMng.requestElement(aElemente(i))) THEN
016          oLayoutMng.createElement(aElemente(i))
017       END IF
018    oLayoutMng.showElement(aElemente(i))
019    NEXT
020    ThisComponent.CurrentController.Sidebar.Visible = True
021    ThisComponent.store()
022 END SUB
```

Die darzustellenden Symbolleisten werden explizit benannt. Ist eine der entsprechenden Symbolleisten nicht für den Layoutmanager vorhanden, so wird sie zuerst über **createElement**

erstellt und danach über **showElement** gezeigt. Deshalb muss das Dokument anschließend abgespeichert werden. Diese Prozedur muss über **Extras → Anpassen → Ereignisse → Dokument wird geschlossen → Symbolleisten_Einblenden** eingebunden werden.

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.

```
001 FUNCTION Fullscreen(boSwitch AS BOOLEAN)
002    DIM oDispatcher AS OBJECT
003    DIM Props(0) AS NEW com.sun.star.beans.PropertyValue
004    oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
005    Props(0).Name = "FullScreen"
006    Props(0).Value = boSwitch
007    oDispatcher.executeDispatch(ThisComponent.CurrentController.Frame,
008       ".uno:FullScreen", "", 0, Props())
009 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.

```
001 SUB Vollbild_ein
002    Fullscreen(true)
003    Symbolleisten_Ausblenden
004 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:

```
001 SUB Vollbild_aus
002    Fullscreen(false)
003    Symbolleisten_Einblenden
004 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.

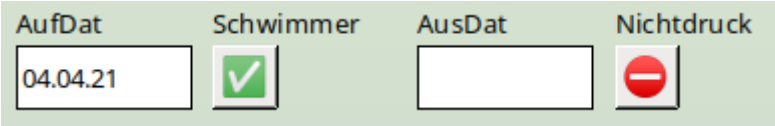
Das folgende Makro wird über **Extras → Anpassen → Ereignisse → Dokument öffnen** gestartet. Dabei ist **Speichern in → Datenbankdatei.odb** zu wählen.

```
001 SUB Formular_Direktstart
002    DIM oDatenquelle AS OBJECT
003    oDatenquelle = ThisDatabaseDocument.CurrentController
004    If NOT (oDatenquelle.isConnected()) THEN
005       oDatenquelle.connect()
006    END IF
007    ThisDatabaseDocument.FormDocuments.getByName("Formularname").open
008    REM alternativ geht auch:
          'oDatenquelle.loadComponent(com.sun.star.sdb.application.DatabaseObject.FORM,
          "Formularname",FALSE)
009 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.

Markierfelder durch Schaltflächen ersetzen

Markierfelder und auch Optionsfelder sind von der Größe und dem Erscheinungsbild her nicht bearbeitbar. Die folgende Lösung erstellt statt Markierfeldern Schaltflächen, die mit einem entsprechenden Symbol versehen sind und wie gewohnt als Markierfelder ansprechbar sind.



Buttons mit Symbolen aus Fonts statt Markierfelder: Vergrößerbar und optisch anpassbar.

Zuerst werden globale Variablen für die beiden Zeichen erstellt, die auf den Buttons abgebildet werden sollen. Die Variablen werden in der Prozedur «BoolStart» mit dem entsprechenden Inhalt versehen, der in dem Beispiel je einem UTF8-Zeichen entspricht.

```
001  GLOBAL stChecked AS STRING
002  GLOBAL stUnChecked AS STRING
```

Damit das Makro nicht speziell auf ein Formular angepasst ist wird der Name des nachgebauten Markierfeldes in einem versteckten Kontrollfeld «hidCheckbox» notiert; bei mehreren Feldern sind diese durch ein Semikolon getrennt. In den Zusatzinformationen jedes einzelnen nachgebauten Markierfeldes steht dann der Name des dazugehörigen Datenfeldes aus der Datenquelle.

Das Einstellen des Wertes des Boolean-Feldes erfolgt beim Wechsel des Datensatzes im Formular. Die Prozedur wird also über **Ereignisse → Nach dem Datensatzwechsel** ausgelöst.

```
001  SUB BoolStart(oEvent AS OBJECT)
002     DIM oForm AS OBJECT
003     DIM oField AS OBJECT
004     DIM stCheckbox AS STRING
005     DIM stCheck AS STRING
006     DIM stValue AS STRING
007     DIM arCheck()
008     oForm = oEvent.Source
```

Damit das Makro nur dann durchläuft, wenn wirklich die entsprechende Checkbox verfügbar ist, wird hier das entsprechende **UnoInterface** abgefragt. Anschließend wird die Variable für «True» und «False» mit dem entsprechenden Zeichen versehen. Schließlich wird aus dem versteckten Feld ausgelesen, welche anderen Felder jetzt Markierfelder darstellen sollen.

```
009     IF hasUnoInterfaces( oForm, "com.sun.star.form.XForm" ) THEN
010        stChecked = "✅"
011        stUnChecked = "⛔"
012        stCheckbox = oForm.getByName("hidCheckbox").HiddenValue
013        arCheck = split(stCheckbox,";")
014        FOR n = LBound(arCheck()) TO UBound(arCheck())
015           oField = oForm.getByName(Trim(arCheck(n)))
016           stCheck = oField.Tag
```

Wenn es sich um einen leeren Datensatz handelt (letzter neuer Datensatz), dann soll der Wert für das Feld 'False' sein. Es wird also kein Markierfeld mit 3 verschiedenen Einstellmöglichkeiten dargestellt.

Ist der ausgelesene Wert aus der Tabelle 'True', dann wird das Markierfeld mit dem entsprechenen Symbol versehen. Für alle anderen Werte wird 'false' angenommen.

```
017           IF oForm.IsRowCountFinal AND oForm.RowCount = 0 THEN
018              stValue = "false"
019           ELSE
020              stValue = oForm.getString(oForm.findColumn(stCheck))
021           END IF
022           IF stValue = "true" THEN
023              oField.Label = stChecked
```

```
024             ELSE
025                oField.Label = stUnChecked
026             END IF
027          NEXT
028       END IF
029    END SUB
```

Der Button wird in der Regel durch die Maus ausgelöst. Damit aber nicht ein Ansteuern über den Tabulator, das auch das **Ereignis → Taste gedrückt** auslöst, das Feld umstellt, muss hier vorher der KeyCode für den Tablulator, 1282, siehe *http://api.libreoffice.org/docs/idl/ref/namespacecom_1_1sun_1_1star_1_1awt_1_1Key.html* herausgefiltert werden. Dies wurde hier in eine separate Prozedur ausgelagert, da die Schaltfläche auch über die Maus ausgelöst werden kann.

```
001  SUB BoolChangeKey(oEvent AS OBJECT)
002     IF oEvent.KeyCode <> 1282 THEN
003        BoolChange(oEvent)
004     END IF
005  END SUB
```

Wenn die Beschriftung des Feldes auf 'stChecked' steht, dann wird sie durch dieses Makro umgestellt auf 'stUnChecked' außerdem wird das Datenfeld auf 'false' eingestellt.

Das Makro wird von der Schaltfläche über **Ereignisse → Taste gedrückt** (Umweg über SUB BoolChangeKey) und über **Ereignisse → Maustaste gedrückt** ausgelöst.

```
001  SUB BoolChange(oEvent AS OBJECT)
002     DIM oField AS OBJECT
003     DIM oForm AS OBJECT
004     oField = oEvent.Source.Model
005     oForm = oField.Parent
006     IF oField.Label = stChecked THEN
007        oField.Label = stUnChecked
008        oForm.updateBoolean(oForm.findColumn(oField.Tag),false)
009     ELSE
010        oField.Label = stChecked
011        oForm.updateBoolean(oForm.findColumn(oField.Tag),true)
012     END IF
013  END SUB
```

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:

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

Da in Makros der SQL-Befehl Text darstellt, müssen die doppelten Anführungszeichen zusätzlich maskiert werden:

```
001  stSQL = "SELECT ""Feld"" FROM ""Tabelle"""
```

MySQL-Abfragen können hingegen anders maskiert werden:

```
001  SELECT `Feld` FROM `Datenbank`.`Tabelle`
```

Durch diese andere Form der Maskierung wird daraus im Makro-Code:

```
001  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:

```
001 SUB CreateTempTable
002    oDatenquelle = thisDatabaseDocument.CurrentController
003    IF NOT (oDatenquelle.isConnected()) THEN oDatenquelle.connect()
004    oVerbindung = oDatenquelle.ActiveConnection()
005    oSQL_Anweisung = oVerbindung.createStatement()
006    stSql = "CREATE TEMPORARY TABLE IF NOT EXISTS `Suchtmp` (`ID` INT PRIMARY KEY,
          `Name` VARCHAR(50))"
007    oSQL_Anweisung.executeUpdate(stSql)
008 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:

```
001    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:

```
001 SUB DeleteFilter
002    oDatasource = thisDatabaseDocument.CurrentController
003    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
004    oConnection = oDatasource.ActiveConnection()
005    oSQL_Command = oConnection.createStatement()
006    stSql = "DELETE FROM `Filter` WHERE `Connection_ID` = CONNECTION_ID()"
007    oSQL_Command.executeUpdate(stSql)
008 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 Ergeb-

nismenge in einer temporären Tabelle, so lässt sich diese temporäre Tabelle als nicht bearbeitbare Informationsquelle nutzen.

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.

```
001 SUB ProcExecute
002    oDatasource = thisDatabaseDocument.CurrentController
003    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
004    oConnection = oDatasource.ActiveConnection()
005    oSQL_Command = oConnection.createStatement()
006    oSql_Command.executeUpdate("CALL `AlleNamen`();")
007 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.

```
001 SUB ProcContentShow
002    oDatasource = thisDatabaseDocument.CurrentController
003    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
004    oConnection = oDatasource.ActiveConnection()
005    oSQL_Command = oConnection.createStatement()
006    oResult = oSql_Command.executeQuery("CALL `AlleNamen`();")
007    stFields = ""
008    FOR i = 1 TO oResult.Columns.Count
009       stFields = stFields + "`" + oResult.Columns.ElementNames(i-1) + "` TINYTEXT,"
010    NEXT
011    stFields = Left(stFields, Len(stFields)-1)
012    stProcedure = "("
013    WHILE oResult.next
014       FOR i = 1 TO oResult.Columns.Count
015          stProcedure = stProcedure + "'" + oResult.getString(i) +"',"
016       NEXT
017       stProcedure =  Left(stProcedure, Len(stProcedure)-1)
018       stProcedure = stProcedure + "),("
019    WEND
020    stProcedure = Left(stProcedure, Len(stProcedure)-2)
021    oSQL_Command.executeUpdate("DROP TEMPORARY TABLE IF EXISTS `TempNamen`")
022    oSQL_Command.executeUpdate("CREATE TEMPORARY TABLE `TempNamen` ("+stFields+")")
023    oSQL_Command.executeUpdate("INSERT INTO `TempNamen` VALUES "+stProcedure+";")
024 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.

PostgreSQL und Makros

Autowertrückgabe mit Returning

Existiert bei PostgreSQL ein AutoWert-Feld, so kann aus diesem Feld mit dem folgenden Befehl der gerade neu erstellte AutoWert ermittelt werden:

```
001 Sub Insert_Returning
002    DIM oDatasource AS OBJECT
003    DIM oConnection AS OBJECT
004    DIM stSql AS STRING
005    DIM oResult AS OBJECT
006    DIM loID AS LONG
007    oDatasource = thisDatabaseDocument.CurrentController
008    IF NOT (oDatasource.isConnected()) THEN oDatasource.connect()
009    oConnection = oDatasource.ActiveConnection()
010    oSQL_Statement = oConnection.createStatement()
011    stSql = "INSERT INTO ""Test1"" (""Neu"") Values('Ich') RETURNING ""ID"""
012    oResult = oSQL_Statement.executeQuery(stSql)
013    oResult.Next
014    loID = oResult.getLong(1)
015 End Sub
```

Der zurückgegebene Schlüsselwert kann nur ausgelesen werden, wenn der Datensatz über **executeQuery** eingefügt wird.

Liegt die Tabelle nicht im Schema «public», dann ist der Name des Schemas mit aufzunehmen:

```
011    stSql = "INSERT INTO ""loffice"".""Test1"" (""Neu"") Values('Ich')
          RETURNING ""ID"""
```

fügt einen Wert in eine Tabelle ein, die in dem selbst erstellten Schema «loffice» liegt.

Datentyp «Array»

Mit PostgreSQL kann einem Feld auch der Datentyp «Array» zugewiesen werden[24]. Dies geht allerdings nur über **Extras → SQL**:

```
001 CREATE TABLE "public"."tbl_Array" (
002    "ID" int4 NOT NULL,
003    "Nachname" varchar(100),
004    "Vornamen" varchar(200)[],
005    PRIMARY KEY ("ID"));
```

In das Feld "Vornamen" können jetzt über eine geschweifte Klammer, getrennt mit Kommas, Array eingegeben werden. Diese Werte können in Abfragen einzeln ausgelesen werden.

```
001 SELECT "Nachname", "Vornamen"[1] FROM "tbl_Array"
```

Dies gibt den ersten Vornamen in der Liste wieder.

Das Einfügen und auslesen von Werten bei diesen Feldern ist mit Makros etwas umständlich. Natürlich funktioniert die direkte Eingabe mit den geschweiften Klammern, aber bei **prepared Statements** hakt es:

```
001    DIM ar
002    stSql = "INSERT INTO ""public"".""tbl_Array"" (""ID"", ""Nachname"",
          ""Vornamen"") VALUES (?, ?, ?)"
003    oSQL_Statement = oConnection.prepareStatement(stSql)
004    oSQL_Statement.setLong(1, 2)
005    oSQL_Statement.setString(2, "Big")
006    ar = array("Will","John","Jack")
```

24 Die Arrayfunktion ist nur beim direkten Treiber so implementiert, dass die entsprechenden Werte auch in der Tabelle direkt eingebbar und sichtbar sind. Der JDBC-Treiber unterstützt Arrays nicht in gleichem Umfang.

```
007     oSQL_Statement.setArray(3, ar)
```

Hier kommt es bei **setArray** direkt zum Crash: *Bug 154464*

Das Feld muss, wenn bereits ein Array vorgesehen ist, nicht über **setArray** mit Inhalt versehen werden, sondern über **setString**:

```
006     ar = array("Will","John","Jack")
007     stAr = "{"
008     FOR i = LBound(ar()) TO UBound(ar())
009        stAr = stAr & ar(i) & ","
010     NEXT
011     stAr = Left(stAr,Len(stAr)-1) & "}"
012     oSQL_Statement.setString(3, stAr)
```

Beim Auslesen der Werte über eine Abfrage in Makros funktioniert die entsprechende Methode **getString** allerdings nicht. Dies würde nur für einen Wert (**"Vornamen"[1]**), nicht aber für das Array funktionieren. Stattdessen müssen die Werte über **getArray** ausgelesen werden:

```
001     DIM ar
002     DIM stAr
003     stSql = "SELECT ""ID"", ""Nachname"", ""Vornamen"" FROM
           ""public"".""tbl_Array"""
004     oResult = oSQL_Statement.executeQuery(stSql)
005     WHILE oResult.Next
006        loID = oResult.getLong(1)
007        stSurname = oResult.getString(2)
008        ar = oResult.getArray(3)
009        stAr = ar.getArray(NULL)
010        FOR i = LBound(stAr) TO UBound(stAr)
011           PRINT stAr(i)
012        NEXT
013     WEND
```

Die Werte aus dem Arrayfeld werden hier zu Demonstrationszwecken lediglich auf dem Bildschirm ausgegeben. Sie können entsprechend anderweitig umgeformt und ausgegeben 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[25] 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.

25 Die Beispieldatenbank «Beispiel_Dialoge.odb» zu den folgenden Kapiteln ist den Beispieldatenbanken für dieses Handbuch beigefügt.

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.

```
001 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.

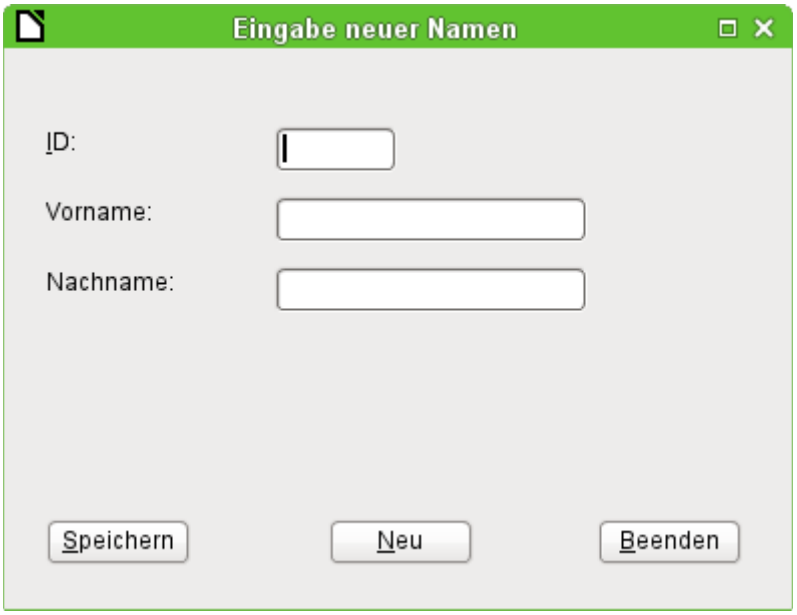
```
001 SUB Dialog0Start
002    DialogLibraries.LoadLibrary("Standard")
003    oDialog0 = createUnoDialog(DialogLibraries.Standard.Dialog0)
004    oDialog0.Execute()
005 END SUB
```

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.

```
001 SUB Dialog0Ende
002    oDialog0.EndExecute()
003 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

```
001 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 «Datenfelder-Leeren» von vorhergehenden Eingaben befreit:

```
001 SUB DatenfelderLeeren
002     oDialog1.getControl("NumericField1").Text = ""
003     oDialog1.getControl("TextField1").Text = ""
004     oDialog1.getControl("TextField2").Text = ""
005 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:

```
001 SUB Daten1Speichern
002     DIM oDatenquelle AS OBJECT
003     DIM oVerbindung AS OBJECT
004     DIM oSQL_Anweisung AS OBJECT
```

```
005     DIM loID AS LONG
006     DIM stVorname AS STRING
007     DIM stNachname AS STRING
008     loID = oDialog1.getControl("NumericField1").Value
009     stVorname = oDialog1.getControl("TextField1").Text
010     stNachname = oDialog1.getControl("TextField2").Text
011     IF loID > 0 AND stNachname <> "" THEN
012        oDatenquelle = thisDatabaseDocument.CurrentController
013        If NOT (oDatenquelle.isConnected()) THEN
014           oDatenquelle.connect()
015        END IF
016        oVerbindung = oDatenquelle.ActiveConnection()
017        oSQL_Anweisung = oVerbindung.createStatement()
018        stSql = "SELECT ""ID"" FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
019        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
020        WHILE oAbfrageergebnis.next
021           MsgBox ("Der Wert für das Feld 'ID' existiert schon",16,
                  "Doppelte Dateneingabe")
022           EXIT SUB
023        WEND
024        stSql = "INSERT INTO ""Name"" (""ID"", ""Vorname"", ""Nachname"")
              VALUES ('"+loID+"','"+stVorname+"','"+stNachname+"')"
025        oSQL_Anweisung.executeUpdate(stSql)
026        DatenfelderLeeren
027     END IF
028 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.

Der 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.

```
001  SUB DatenLaden(loID AS LONG)
002     DIM oDatenquelle AS OBJECT
003     DIM oVerbindung AS OBJECT
004     DIM oSQL_Anweisung AS OBJECT
005     DIM stVorname AS STRING
006     DIM stNachname AS STRING
007     DIM loRow AS LONG
008     DIM loRowMax AS LONG
009     DIM inStart AS INTEGER
010     oDatenquelle = thisDatabaseDocument.CurrentController
011     If NOT (oDatenquelle.isConnected()) THEN
012        oDatenquelle.connect()
013     END IF
014     oVerbindung = oDatenquelle.ActiveConnection()
015     oSQL_Anweisung = oVerbindung.createStatement()
016     IF loID < 1 THEN
017        stSql = "SELECT MIN(""ID"") FROM ""Name"""
018        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
019        WHILE oAbfrageergebnis.next
020           loID = oAbfrageergebnis.getInt(1)
021        WEND
```

```
022         inStart = 1
023     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.

```
024     IF loID > 0 THEN
025         stSql = "SELECT * FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
026         oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
027         WHILE oAbfrageergebnis.next
028             loID = oAbfrageergebnis.getInt(1)
029             stVorname = oAbfrageergebnis.getString(2)
030             stNachname = oAbfrageergebnis.getString(3)
031         WEND
032         stSql = "SELECT COUNT(""ID"") FROM ""Name"""
033         oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
034         WHILE oAbfrageergebnis.next
035             loRowMax = oAbfrageergebnis.getInt(1)
036         WEND
037         stSql = "SELECT COUNT(""ID"") FROM ""Name"" WHERE ""ID"" <= '"+loID+"'"
038         oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
039         WHILE oAbfrageergebnis.next
040             loRow = oAbfrageergebnis.getInt(1)
041         WEND
042         oDialog2.getControl("NumericField1").Value = loID
043         oDialog2.getControl("TextField1").Text = stVorname
044         oDialog2.getControl("TextField2").Text = stNachname
045     END IF
046     oDialog2.getControl("NumericField2").Value = loRow
047     oDialog2.getControl("NumericField3").Value = loRowMax
048     IF loRow = 1 THEN
049         ' Vorheriger Datensatz
050         oDialog2.getControl("CommandButton4").Model.enabled = False
051     ELSE
052         oDialog2.getControl("CommandButton4").Model.enabled = True
053     END IF
054     IF loRow <= loRowMax THEN
055         ' Nächster Datensatz | Neuer Datensatz | Löschen
056         oDialog2.getControl("CommandButton5").Model.enabled = True
057         oDialog2.getControl("CommandButton2").Model.enabled = True
058         oDialog2.getControl("CommandButton6").Model.enabled = True
059     ELSE
060         oDialog2.getControl("CommandButton5").Model.enabled = False
061         oDialog2.getControl("CommandButton2").Model.enabled = False
062         oDialog2.getControl("CommandButton6").Model.enabled = False
063     END IF
064     IF inStart = 1 THEN
065         oDialog2.Execute()
066     END IF
067 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, 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.

```
001 SUB vorherigerDatensatz
002    DIM loID AS LONG
003    DIM loIDneu AS LONG
004    loID = oDialog2.getControl("NumericField1").Value
005    oDatenquelle = thisDatabaseDocument.CurrentController
006    If NOT (oDatenquelle.isConnected()) THEN
007       oDatenquelle.connect()
008    END IF
009    oVerbindung = oDatenquelle.ActiveConnection()
010    oSQL_Anweisung = oVerbindung.createStatement()
011    IF loID < 1 THEN
012       stSql = "SELECT MAX(""ID"") FROM ""Name"""
013    ELSE
014       stSql = "SELECT MAX(""ID"") FROM ""Name"" WHERE ""ID"" < '"+loID+"'"
015    END IF
016    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
017    WHILE oAbfrageergebnis.next
018       loIDneu = oAbfrageergebnis.getInt(1)
019    WEND
020    IF loIDneu > 0 THEN
021       DatenLaden(loIDneu)
022    END IF
023 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.

```
001  SUB naechsterDatensatz
002     DIM loID AS LONG
003     DIM loIDneu AS LONG
004     loID = oDialog2.getControl("NumericField1").Value
005     oDatenquelle = thisDatabaseDocument.CurrentController
006     If NOT (oDatenquelle.isConnected()) THEN
007        oDatenquelle.connect()
008     END IF
009     oVerbindung = oDatenquelle.ActiveConnection()
010     oSQL_Anweisung = oVerbindung.createStatement()
011     stSql = "SELECT MIN(""ID"") FROM ""Name"" WHERE ""ID"" > '"+loID+"'"
012     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
013     WHILE oAbfrageergebnis.next
014        loIDneu = oAbfrageergebnis.getInt(1)
015     WEND
016     IF loIDneu > 0 THEN
017        DatenLaden(loIDneu)
018     ELSE
019        Datenfelder2Leeren
020     END IF
021  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.

```
001  SUB Datenfelder2Leeren
002     loRowMax = oDialog2.getControl("NumericField3").Value
003     oDialog2.getControl("NumericField1").Text = ""
004     oDialog2.getControl("TextField1").Text = ""
005     oDialog2.getControl("TextField2").Text = ""
006     oDialog2.getControl("NumericField2").Value = loRowMax + 1
007     oDialog2.getControl("CommandButton4").Model.enabled = True   ' Vorh. Datensatz
008     oDialog2.getControl("CommandButton5").Model.enabled = False  ' Nächster Datens.
009     oDialog2.getControl("CommandButton2").Model.enabled = False  ' Neuer Datensatz
010     oDialog2.getControl("CommandButton6").Model.enabled = False  ' Löschen
011  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.

```
001  SUB Daten2Speichern(oEvent AS OBJECT)
002     DIM oDatenquelle AS OBJECT
```

```
003     DIM oVerbindung AS OBJECT
004     DIM oSQL_Anweisung AS OBJECT
005     DIM oDlg AS OBJECT
006     DIM loID AS LONG
007     DIM stVorname AS STRING
008     DIM stNachname AS STRING
009     DIM inMsg AS INTEGER
010     DIM loRow AS LONG
011     DIM loRowMax AS LONG
012     DIM stSql AS STRING
013     oDlg = oEvent.Source.getContext()
014     loID = oDlg.getControl("NumericField1").Value
015     stVorname = oDlg.getControl("TextField1").Text
016     stNachname = oDlg.getControl("TextField2").Text
017     IF loID > 0 AND stNachname <> "" THEN
018        oDatenquelle = thisDatabaseDocument.CurrentController
019        If NOT (oDatenquelle.isConnected()) THEN
020           oDatenquelle.connect()
021        END IF
022        oVerbindung = oDatenquelle.ActiveConnection()
023        oSQL_Anweisung = oVerbindung.createStatement()
024        loRow = oDlg.getControl("NumericField2").Value
025        loRowMax = oDlg.getControl("NumericField3").Value
026        IF loRowMax < loRow THEN
027           IF loRowMax > 0 THEN
028              stSql = "SELECT ""ID"" FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
029              oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
030              WHILE oAbfrageergebnis.next
031                 inMsg = MsgBox ("Der Wert für das Feld 'ID' existiert schon." &
                       CHR(13) & "Soll der Datensatz überschrieben werden?",20,
                       "Doppelte Dateneingabe")
032                 IF inMsg = 6 THEN
033                    stSql = "UPDATE ""Name"" SET ""Vorname""='"+stVorname+"',
                          ""Nachname""='"+stNachname+"' WHERE ""ID"" = '"+loID+"'"
034                    oSQL_Anweisung.executeUpdate(stSql)
035                    DatenLaden(loID) ' Beim Update wurde ein bestehender Datensatz
                          überschrieben. Neueinlasen zur Korrektur der Datensatzzahlen
036                 END IF
037                 EXIT SUB
038              WEND
039           END IF
040           stSql = "INSERT INTO ""Name"" (""ID"", ""Vorname"", ""Nachname"") VALUES
                 ('"+loID+"','"+stVorname+"','"+stNachname+"')"
041           oSQL_Anweisung.executeUpdate(stSql)
042           oDlg.getControl("NumericField3").Value = loRowMax + 1
                 ' Nach dem Insert existiert ein Datensatz mehr
043           Datenfelder2Leeren
                 ' Nach einem Insert wird grundsätzlich zum nächsten Insert geschaltet
044        ELSE
045           stSql = "UPDATE ""Name"" SET ""Vorname""='"+stVorname+"',
                 ""Nachname""='"+stNachname+"' WHERE ""ID"" = '"+loID+"'"
046           oSQL_Anweisung.executeUpdate(stSql)
047        END IF
048     END IF
049  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.

```
001  SUB DatenLoeschen(oEvent AS OBJECT)
002     DIM oDatenquelle AS OBJECT
```

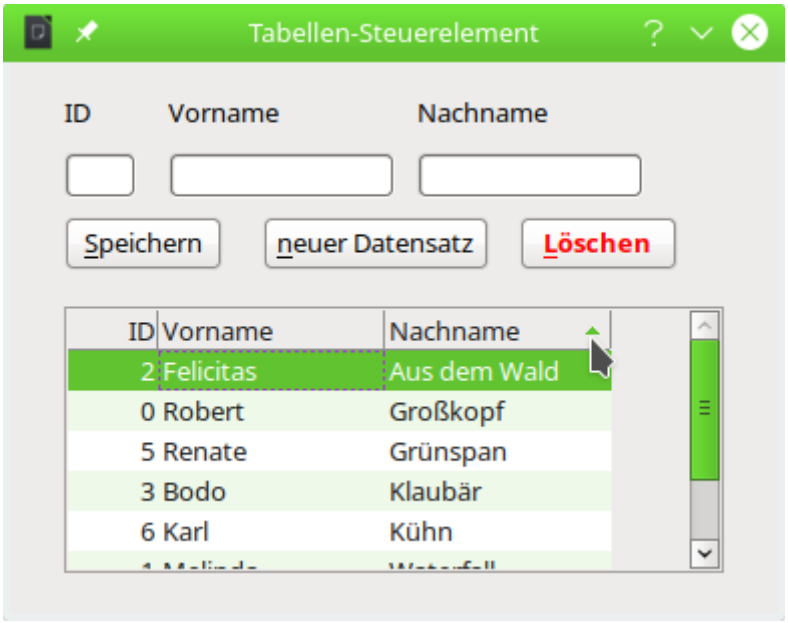
```
003     DIM oVerbindung AS OBJECT
004     DIM oSQL_Anweisung AS OBJECT
005     DIM oDlg AS OBJECT
006     DIM loID AS LONG
007     oDlg = oEvent.Source.getContext()
008     loID = oDlg.getControl("NumericField1").Value
009     IF loID > 0 THEN
010        inMsg = MsgBox ("Soll der Datensatz wirklich gelöscht werden?",20,
           "Löschen eines Datensatzes")
011        IF inMsg = 6 THEN
012           oDatenquelle = thisDatabaseDocument.CurrentController
013           If NOT (oDatenquelle.isConnected()) THEN
014              oDatenquelle.connect()
015           END IF
016           oVerbindung = oDatenquelle.ActiveConnection()
017           oSQL_Anweisung = oVerbindung.createStatement()
018           stSql = "DELETE FROM ""Name"" WHERE ""ID"" = '"+loID+"'"
019           oSQL_Anweisung.executeUpdate(stSql)
020           loRowMax = oDlg.getControl("NumericField3").Value
021           oDlg.getControl("NumericField3").Value = loRowMax - 1
022           naechsterDatensatz
023        END IF
024     ELSE
025        MsgBox ("Kein Datensatz gelöscht." & CHR(13) &
           "Es fehlt eine Datensatzauswahl.",64,"Löschung nicht möglich")
026     END IF
027  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.

Dialog zum Bearbeiten von Daten aus einer Tabellenübersicht

Zusammen mit dem Tabellen-Steuerelement der Dialoge ist es möglich, aus einer vorhandenen Datenmenge Datensätze auszuwählen und zu bearbeiten, neue Datensätze einzufügen oder auch vorhandene Daten zu löschen. Das Tabellen-Steuerelement dient dabei zur Auswahl der Datensätze. Die Bearbeitung erfolgt wie bei den vorhergehenden Dialogen über einfache For-

mularfelder. Um die Verwaltung der Daten einfacher zu machen ist bei der verwendeten Tabelle ein automatisch erstellter Primärschlüssel verwendet worden.



Das Tabellen-Steuerelement bietet neben den in wechselnden Farben erscheinenden Tabellenzeilen auch die Möglichkeit, die Daten nach den Tabellenköpfen zu sortieren. Im Screenshot ist die aktuelle Sortierung für das Feld «Nachname» durch ein kleines grünes Dreieck zu erkennen.

```
001 DIM oDialog3 AS OBJECT
```

```
001 SUB Dialog3Start
002    DialogLibraries.LoadLibrary("Standard")
003    oDialog3 = createUnoDialog(DialogLibraries.Standard.Dialog3)
004    GridDatenZeigen
005    oDialog3.Execute()
006 END SUB
```

```
001 SUB Dialog3Ende
002    oDialog3.EndExecute()
003 END SUB
```

Wie bei den anderen Dialogen muss die Variable für den Dialog außerhalb der Prozeduren notiert werden, damit sie in dem gesamten Modul verfügbar ist. Über «Dialog3Start» wird der Dialog gestartet. Wichtig ist für das Tabellen-Kontrollfeld, dass die Prozedur «GridDatenZeigen» vor der Ausführung des Dialogs abläuft. Sonst bleibt die Tabelle leer.

Die Prozedur zum Beenden des Dialogs ist nur notwendig, wenn ein Button zum Beenden mit eingebaut werden soll. Das Schließen des Dialogs über das X erfolgt unabhängig von der Prozedur.

```
001 SUB GridDatenZeigen
002    DIM oGrid AS OBJECT
003    DIM oColumnModel AS OBJECT
004    DIM oColumn1 AS OBJECT
005    DIM oColumn2 AS OBJECT
006    DIM oColumn3 AS OBJECT
007    DIM oDataModel AS OBJECT
008    DIM oDatenquelle AS OBJECT
009    DIM oVerbindung AS OBJECT
010    DIM oSQL_Anweisung AS OBJECT
011    DIM stSql AS STRING
012    DIM oAbfrageergebnis AS OBJECT
013    DIM l AS LONG
014    DIM stID AS STRING
```

```
015     DIM stVorname AS STRING
016     DIM stNachname AS STRING
017     oGrid = oDialog3.Model.getByName("GridControl1")
018     oColumnModel = oGrid.ColumnModel
```

Nach Deklaration der Variablen wird auf die Tabelle Zugegriffen. Zuerst werden die Spalten der Tabelle erstellt. Die folgenden Einstellungen sind für jede Spalte notwendig. Sie können natürlich auch platzsparender über Arrays erzeugt werden.

```
019     oColumn1 = createUnoService("com.sun.star.awt.grid.GridColumn")
020     oColumn1.Title = "ID"
021     oColumn1.ColumnWidth = 20
022     oColumn1.HorizontalAlign = 2
023     oColumn1.Flexibility = False
```

Die Ausrichtung in **HorizontalAlign** wird über die Zuweisung von Werten geregelt. '0' steht für linksbündig, '1' für zentriert und '2' für rechtsbündig. Wird die **Flexibility** nicht auf **False** gesetzt, dann wird die Spaltenbreite von **ColumnWidth** nicht richtig übertragen. Die erste Spalte wird durch die Automatik dann viel zu breit.

```
024     oColumn2 = createUnoService("com.sun.star.awt.grid.GridColumn")
025     oColumn2.Title = "Vorname"
026     oColumn2.ColumnWidth = 50
027     oColumn2.HorizontalAlign = 0
028     oColumn2.Flexibility = False
029     oColumn3 = createUnoService("com.sun.star.awt.grid.GridColumn")
030     oColumn3.Title = "Nachname"
031     oColumn3.ColumnWidth = 50
032     oColumn3.HorizontalAlign = 0
033     oColumn3.Flexibility = False
034     oColumnModel.AddColumn(oColumn1)
035     oColumnModel.AddColumn(oColumn2)
036     oColumnModel.AddColumn(oColumn3)
```

Nachdem die Spalten mit den Benennungen erstellt wurden, werden die Daten hinzugefügt. Dazu wird zuerst die Datenbankverbindung überprüft und gegebenenfalls erzeugt. Die gesamten Daten werden abgefragt und Datensatz für Datensatz über **addRow** hinzugefügt. In der Klammer von **addRow** steht zuerst die Datensatznummer und danach ein Array mit den Inhalten des Datensatzes.

```
037     oDataModel = oGrid.GridDataModel
038     oDatenquelle = thisDatabaseDocument.CurrentController
039     If NOT (oDatenquelle.isConnected()) THEN
040        oDatenquelle.connect()
041     END IF
042     oVerbindung = oDatenquelle.ActiveConnection()
043     oSQL_Anweisung = oVerbindung.createStatement()
044     stSql = "SELECT * FROM ""Name_ID_Autowert"" "
045     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
046     l = 1
047     WHILE oAbfrageergebnis.next
048        stID = oAbfrageergebnis.getString(1)
049        stVorname = oAbfrageergebnis.getString(2)
050        stNachname = oAbfrageergebnis.getString(3)
051        oDataModel.addRow (l, Array(stID, stVorname, stNachname))
052        l = l + 1
053     WEND
054  END SUB
```

Mit der Prozedur **GridRow** werden die Daten aus dem markierten Datensatz in die Formularfelder oberhalb der Tabelle übertragen. Dadurch können dann Daten geändert oder gelöscht werden.

```
001  SUB GridRow(oEvent AS OBJECT)
002     DIM oGrid AS OBJECT
003     DIM loRow AS LONG
004     DIM oDataModel AS OBJECT
005     DIM stData AS STRING
```

```
006     DIM oDatenquelle AS OBJECT
007     DIM oVerbindung AS OBJECT
008     DIM oSQL_Anweisung AS OBJECT
009     DIM stSql AS STRING
010     DIM oAbfrageergebnis AS OBJECT
011     DIM loID AS LONG
012     DIM stVorname AS STRING
013     DIM stNachname AS STRING
014     oGrid = oEvent.Source
015     IF oGrid.hasSelectedRows THEN
```

Die Klicks auf das Tabellen-Kontrollfeld lösen dieses Makro aus. Sie landen nicht unbedingt auf irgendeinem Datensatz sondern z.B. auch auf den Tabellenköpfen. Wenn der Hintergrund oder die Tabellenköpfe angeklickt wurden können keine Daten ausgelesen werden. Deswegen muss erst einmal klar sein, ob eine Zeile ausgewählt wurde.

Aus dem markierten Datensatz wird das erste Feld '0' über **getCellData** ausgewählt. In ihm ist in diesem Falle der Primärschlüssel gespeichert. Dadurch kann der Datensatz eindeutig erkannt werden. Neben den Inhalten aus der Datenbank wird auch die Zeilennummer aus dem Tabellen-Kontrollfeld ausgelesen und in einem versteckten Formularfeld zwischengespeichert.

```
016         loRow = oGrid.CurrentRow()
017         oDataModel = oGrid.Model.GridDataModel
018         stData = oDataModel.getCellData(0, loRow)
019         oDatenquelle = thisDatabaseDocument.CurrentController
020         If NOT (oDatenquelle.isConnected()) THEN
021            oDatenquelle.connect()
022         END IF
023         oVerbindung = oDatenquelle.ActiveConnection()
024         oSQL_Anweisung = oVerbindung.createStatement()
025         stSql = "SELECT * FROM ""Name_ID_Autowert"" WHERE ""ID"" = '"+stData+"'"
026         oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
027         WHILE oAbfrageergebnis.next
028            loID = oAbfrageergebnis.getLong(1)
029            stVorname = oAbfrageergebnis.getString(2)
030            stNachname = oAbfrageergebnis.getString(3)
031         WEND
032         oDialog3.getControl("NumericField1").Value = loID
033         oDialog3.getControl("TextField1").Text = stVorname
034         oDialog3.getControl("TextField2").Text = stNachname
035         oDialog3.getControl("Zeilennummer").Value = loRow
036     END IF
037 END SUB
```

Wird der Speicher-Button betätigt, so läuft die folgende Prozedur ab.

```
001 SUB GridDatenSpeichern
002     DIM stID AS STRING
003     DIM stVorname AS STRING
004     DIM stNachname AS STRING
005     DIM loRow AS LONG
006     DIM oDatenquelle AS OBJECT
007     DIM oVerbindung AS OBJECT
008     DIM oSQL_Anweisung AS OBJECT
009     DIM stSql AS STRING
010     DIM oAbfrageergebnis AS OBJECT
011     DIM oGridData AS OBJECT
012     DIM l AS LONG
013     stID = oDialog3.getControl("NumericField1").Text
014     stVorname = oDialog3.getControl("TextField1").Text
015     stNachname = oDialog3.getControl("TextField2").Text
016     loRow = oDialog3.getControl("Zeilennummer").Value
```

Die Einträge aus den Formularfeldern werden ausgelesen. Dabei erfolgt das Auslesen auch des numerischen Feldes für die 'ID' als Text. So kann auf ein leeres Feld anschließend besser überprüft werden. Bei einem leeren Feld für die 'ID' handelt es sich um einen neuen Datensatz. Bei einem belegten Feld um eine Änderung des Datensatzes.

```
017     oDatenquelle = thisDatabaseDocument.CurrentController
018     If NOT (oDatenquelle.isConnected()) THEN
019        oDatenquelle.connect()
020     END IF
021     oVerbindung = oDatenquelle.ActiveConnection()
022     oSQL_Anweisung = oVerbindung.createStatement()
023     IF stID <> "" THEN
024        stSql = "UPDATE ""Name_ID_Autowert"" SET ""Vorname"" = '"+stVorname+"',
              ""Nachname"" = '"+stNachname+"' WHERE ID = '"+stID+"'"
025     ELSE
026        stSql = "INSERT INTO ""Name_ID_Autowert"" (""Vorname"", ""Nachname"") VALUES
              ('"+stVorname+"', '"+stNachname+"')"
027     END IF
028     oSQL_Anweisung.executeUpdate(stSql)
```

Bei Firebird muss der Code für den INSERT angepasst werden. Statt der einfachen INSERT-Anweisung ist der folgende Weg nötig:

```
026        stSql = "SELECT NEXT VALUE FOR RDB$1 FROM RDB$DATABASE"
           oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
           oAbfrageergebnis.next
           loID = oAbfrageergebnis.getLong(1)
           stSql = "INSERT INTO ""Name_ID_Autowert"" (""ID"", ""Vorname"", ""Nachname"")
              VALUES ('"+stID+"','"+stVorname+"', '"+stNachname+"')"
```

Der Name des Generator für die ID wird in einer Abfrage mit direktem SQL über

```
001 SELECT RDB$FIELD_NAME, RDB$RELATION_NAME, RDB$GENERATOR_NAME
    FROM RDB$RELATION_FIELDS
002 WHERE RDB$GENERATOR_NAME IS NOT NULL
```

ermittelt. Über den Namen des Generators für die ID wird dann der nächste freie Wert des Generators abgefragt und in der Variablen **loID** zwischengespeichert. Dieser Wert ist damit vergeben und kann in dem folgenden **INSERT** für die "ID" genutzt werden. Dies ist zur Zeit die sicherste Variante, da **RETURNING** nicht funktioniert und die anschließende Abfrage des höchsten Wertes für "ID" nur dann sicher ist, wenn der Generator nicht zurückgesetzt wurde und das System kein Mehrbenutzersystem ist. In Mehrbenutzersystemen würde gegebenenfalls sonst die "ID" eines anderen eingefügten Datensatzes abgefragt.

Nach der Datensatzänderung oder dem Einfügen eines neuen Datensatzes muss auch das Tabellen-Kontrollfeld mit den neuen Daten versorgt werden. Bei der Änderung müssen die Felder über **updateCellData** mit den neuen Daten versorgt werden. Die erste Zahl in der Klammer steht hier für die Spalte, die zweite Zahl in der Klammer für den Datensatz.

```
029     oGridData = oDialog3.Model.getByName("GridControl1").GridDataModel
030     IF stID <> "" THEN
031        oGridData.updateCellData(1,loRow,stVorname)
032        oGridData.updateCellData(2,loRow,stNachname)
033     ELSE
```

Beim Einfügen eines neuen Datensatzes muss zuerst ermittelt werden, wie der automatisch erstellte Primärschlüsselwert lautet. In diesem Fall wird gleich der gesamte Datensatz noch einmal eingelesen, so dass in dem Tabellen-Kontrollfeld auf jeden Fall das steht, was auch in der Datenbank erscheint.

Über **addRow** wird der Datensatz dem Tabellen-Kontrollfeld hinzugefügt. Anschließend werden noch die gerade ermittelten Werte für die 'ID' und auch die Zeilennummer in die Formularfelder eingefügt. Die Zeilennummer entspricht dabei der Gesamtzahl der Datensätze, da der neue Datensatz als letzte Zeile in das Tabellen-Kontrollfeld aufgenommen wird.

```
034        stSql = "SELECT ""Name_ID_Autowert"".*, (SELECT COUNT(""ID"") FROM
              ""Name_ID_Autowert"") FROM ""Name_ID_Autowert"" WHERE ""ID"" = IDENTITY()"
```

In Firebird wird hier **IDENTITY()** durch **'"+loID+"'** ersetzt. Die Variable wurde ja bereits ermittelt.

```
035        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
036        WHILE oAbfrageergebnis.next
037           stID = oAbfrageergebnis.getString(1)
```

```
038          stVorname = oAbfrageergebnis.getString(2)
039          stNachname = oAbfrageergebnis.getString(3)
040          l = oAbfrageergebnis.getLong(4)
041       WEND
042       oGridData.addRow (l, Array(stID, stVorname, stNachname))
043       oDialog3.getControl("NumericField1").Value = stID
044       oDialog3.getControl("Zeilennummer").Value = l
045    END IF
046 END SUB
```

Um neue Daten einzufügen müssen die Daten aus den Eingabefeldern entleert werden. Dies ist besonders wichtig bei dem Feld für den Primärschlüssel (das nicht beschreibbar ist) und dem versteckten Feld für die Zeilennummer.

```
001 SUB GridDatenNeu
002    oDialog3.getControl("NumericField1").Text = ""
003    oDialog3.getControl("TextField1").Text = ""
004    oDialog3.getControl("TextField2").Text = ""
005    oDialog3.getControl("Zeilennummer").Text = ""
006    oDialog3.getControl("GridControl1").deselectAllRows
007 END SUB
```

Sämtliche Eingabefelder werden geleert. Damit nicht beim nächsten Klick, z. B. zur Sortierung der markierte Datensatz in der Eingabezeile auftaucht, wird für alle Zeilen in dem Tabellen-Steuerelement die Selektion aufgehoben.

Die folgende Prozedur zum Löschen ist bereits im 2. Dialog ähnlich enthalten. Es können nur die Daten gelöscht werden, die in den Formularfeldern angezeigt werden. Dafür wird der versteckte Wert des Feldes für die Zeilennummer ausgelesen. Nach einer Kontrollabfrage wird dann der entsprechende SQL-Befehl ausgeführt und die Zeile über **removeRow** aus dem Tabellen-Kontrollfeld entfernt.

```
001 SUB GridDatenLoeschen
002    DIM loID AS LONG
003    DIM loRow AS LONG
004    DIM oGridData AS OBJECT
005    DIM inMsg AS INTEGER
006    DIM oDatenquelle AS OBJECT
007    DIM oVerbindung AS OBJECT
008    DIM oSQL_Anweisung AS OBJECT
009    DIM stSql AS STRING
010    loID = oDialog3.getControl("NumericField1").Value
011    loRow = oDialog3.getControl("Zeilennummer").Value
012    oGridData = oDialog3.Model.getByName("GridControl1").GridDataModel
013    IF loRow > 0 THEN
014       inMsg = MsgBox ("Soll der Datensatz wirklich gelöscht werden?",20,
          "Löschen eines Datensatzes")
015       IF inMsg = 6 THEN
016          oDatenquelle = thisDatabaseDocument.CurrentController
017          If NOT (oDatenquelle.isConnected()) THEN
018             oDatenquelle.connect()
019          END IF
020          oVerbindung = oDatenquelle.ActiveConnection()
021          oSQL_Anweisung = oVerbindung.createStatement()
022          stSql = "DELETE FROM ""Name_ID_Autowert"" WHERE ""ID"" = '"+loID+"'"
023          oSQL_Anweisung.executeUpdate(stSql)
024          oGridData.removeRow(loRow)
025       END IF
026    ELSE
027       MsgBox ("Kein Datensatz gelöscht." & CHR(13) &
          "Es fehlt eine Datensatzauswahl.",64,"Löschung nicht möglich")
028    END IF
029    GridDatenNeu
030 END SUB
```

Fortschrittsbalken für den Ablauf mehrerer Prozeduren

Wird für den Ablauf mehrerer Prozeduren hintereinander eine längere Zeit benötigt, so neigt der Nutzer / die Nutzerin schnell dazu, von einem «Hängen» des Systems auszugehen. Da bietet es sich an klar zu zeigen, dass der Rechner noch beschäftigt ist. Der folgende Dialog wurde für das Einlesen von Daten in eine im Netz befindliche PostgreSQL-Datenbank genutzt.

Zwei Beschriftungsfelder und ein Fortschrittsbalken ergeben den Aufbau des gesamten Dialogs. Der Dialog zeigt über die Vorgangsnummer und den Fortschrittsbalken an, wie weit der Import bisher fortgeschritten ist.

```
001 GLOBAL oDialog1 AS OBJECT
```

Zuerst wird eine allgemeingültige Variable für den Dialog erstellt. Dadurch kann von der Startprozedur aus der Code für die Änderung der Anzeige ausgelagert werden.

```
001 SUB Dialog1Start
002    DIM inWidth AS INTEGER
003    DIM inHeight AS INTEGER
004    DIM inx AS INTEGER
005    DIM iny AS INTEGER
006    DialogLibraries.LoadLibrary("Standard")
007    oDialog1 = createUnoDialog(DialogLibraries.Standard.D_Import)
008    oFrame = ThisComponent.CurrentController.Frame
009    oWin = oFrame.getContainerWindow()
010    inWidth = 315
011    inHeight = 100
012    inx = Int((oWin.Size.Width - inWidth) / 2)
013    iny = Int((oWin.Size.Height - inHeight) / 2)
014    oDialog1.setPosSize(inx, iny, inWidth, inHeight, 15)
015    oDialog1.setVisible(true)
016    FOR i = 1 TO 13
017       stLabel = "Vorgang " & i & " von 13"
018       Progress(i*100/13, stLabel)
019       SELECT CASE i
020          CASE 1
021             Import_Gemeinde
022          CASE 2
023             Import_Ort
024          CASE 3
                ...
025          CASE 13
026             Import_Eigentuemer
027       END SELECT
028    NEXT
029    oDialog1.dispose()
030 END SUB
```

Zuerst werden in der Prozedur **Dialog1Start** die Variablen deklariert. Hier sind nicht alle Variablen aufgeführt, die im Weiteren Verwendung finden.

Die Zeilen 8 bis 14 dienen dazu, den Dialog auf dem Bildschirm zu zentrieren. Dazu wird die Größenvorgabe des Dialogs mit **inWidth** und **inHeight** sowie die Größenvorgabe des verfügbaren Platzes für das Fenster (**oWin.Size.Width** und **oWin.Size.Height**) benötigt. Der Parameter '15' bei **oDialog1.setPosSize** besagt, dass sowohl die Position als auch die Größe des Dialogs geändert werden.

In Zeile 15 wird der Dialog mit **oDialog1.setVisible(true)** sichtbar geschaltet. Würde hier mit **Execute** der Dialog gestartet, so könnten keine weiteren Prozeduren ablaufen.

Zeile 17 und 18 in der Schleife, die ab Zeile 16 beginnt beeinflussen das Erscheinungsbild des Beschriftungsfeldes direkt über dem Fortschrittsbalken sowie die Länge des Fortschrittsbalkens selbst. Die Änderung dieses Erscheinungsbildes ist in die Prozedur Progress ausgelagert:

```
001  SUB Progress(doVal AS DOUBLE, stLabel AS STRING)
002     oDialog1.getControl("ProgressBar").setValue(doVal)
003     oDialog1.getControl("lblProgressBar").Text = stLabel
004  END SUB
```

Der Fortschrittsbalken hat über das Dialogdesign die Bezeichnung "ProgressBar" erhalten. Der Balken wurde im Design für 100 Einheiten ausgelegt. Da insgesamt 13 Prozeduren nacheinander aufgerufen werden wurde entsprechend 100/13 als die Länge für den Ablauf der ersten Prozedur übernommen.

Das Beschriftungsfeld ist über das Dialogdesign mit "lblProgressBar" ansprechbar. Hier wird nur jeweils die Anzeige der aktuellen Vorgangsnummer geändert.

Mit jedem Schleifendurchgang ändert sich die Variable **i**. Entsprechend tritt ein anderer **SELECT CASE** ein. Die Prozeduren, die ab Zeile 20 aufgerufen werden, sind also schlicht durchnummeriert über **CASE 1**, **CASE 2** usw. Die Schleife springt zum nächsten Wert, wenn die Prozedur abgelaufen ist.

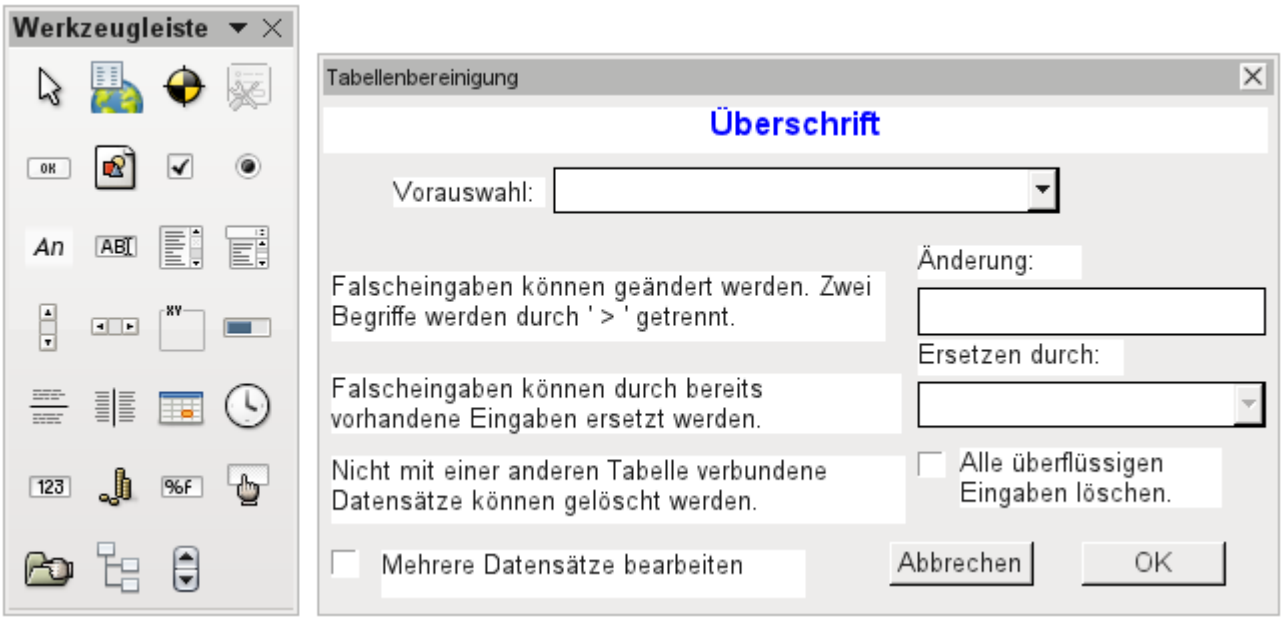
Mit **oDialog1.dispose()** in Zeile 48 wird schließlich der Dialog wieder abgeschaltet. Jetzt könnte noch eine **Messagebox** den erfolgreichen Ablauf aller Prozeduren anzeigen.

Fehleinträge von Tabellen mit Hilfe eines Dialogs bereinigen

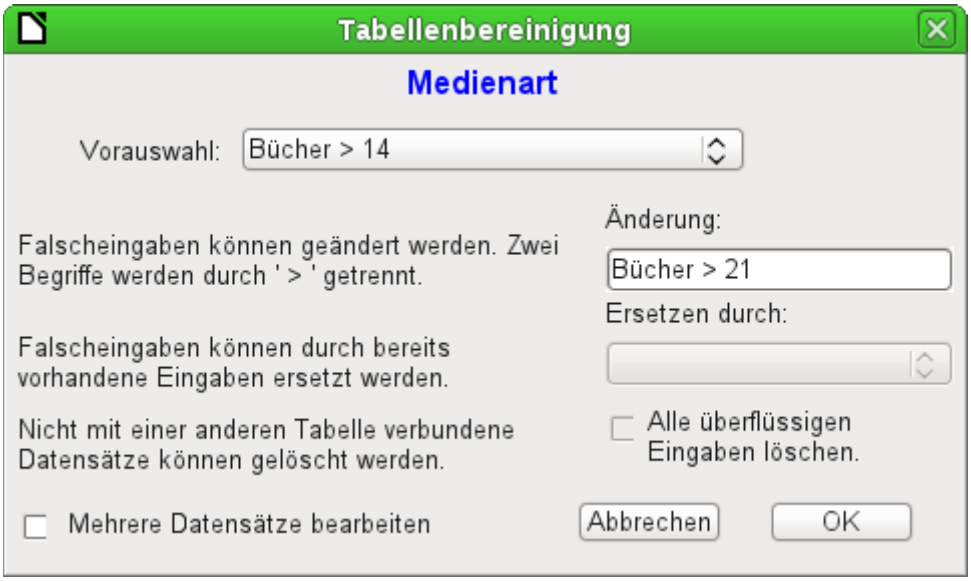
Fehleingaben in Feldern fallen häufig erst später auf. Manchmal müssen auch gleich mehrere Datensätze mit der gleichen Eingabe auf einmal geändert werden. Dies ist in der normalen Tabellenansicht umständlich, je mehr Änderungen vorgenommen werden müssen, da für jeden Datensatz einzeln eine Eingabe erforderlich 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.

```
001 SUB Tabellenbereinigung(oEvent AS OBJECT)
```

Das Makro soll über Einträge im Bereich **Zusatzinformationen** des jeweiligen Buttons gestartet werden.

```
002 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.

```
003     DIM aFremdTabellen(0, 0 to 1)
004     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.

```
005     stTag = oEvent.Source.Model.Tag
006     aTabelle() = Split(stTag, ", ")
007     FOR i = LBound(aTabelle()) TO UBound(aTabelle())
008        aTabelle(i) = trim(aTabelle(i))
009     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.

```
010     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.

```
011     inZaehler = 0
012     stFremdIDTab1Tab2 = "ID"
013     stFremdIDTab2Tab1 = "ID"
014     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.

```
015     oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
016     WHILE oAbfrageergebnis.next
017        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.

```
018        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".

```
019        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.

```
020         IF UBound(aTabelle()) = 8 THEN
021            IF aTabelle(8) = aFremdTabellen(inZaehler,0) THEN
022               stFremdIDTab2Tab1 = aFremdTabellen(inZaehler,1)
023               stNebentabelle = aTabelle(8)
024            END IF
025         END IF
026         inZaehler = inZaehler + 1
```

Da eventuell noch weitere Werte auszulesen sind, erfolgt eine Erweiterung des Zählers zur Neudimensionierung des Arrays. Anschließend wird die Schleife beendet.

```
027      WEND
```

Existiert im Aufruf der Prozedur ein zweiter Tabellenname, so wird die gleiche Abfrage jetzt mit dem zweiten Tabellennamen gestartet:

```
028      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.

```
029         IF aTabelle(5) = aFremdTabellen2(inZaehler,0) THEN
030            stFremdIDTab1Tab2 = aFremdTabellen2(inZaehler,1)
031         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.

```
032      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.

```
033         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.

```
034            stSql = "SELECT """ + aTabelle(6) + """ FROM """ + aTabelle(5)
035               + """ ORDER BY """ + aTabelle(6) + """"
036         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.

```
037            stSql = "SELECT """ + aTabelle(6) + """||' > '||""" + aTabelle(7)
038               + """ FROM """ + aTabelle(5) + """ ORDER BY """ + aTabelle(6) + """"
039         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.

```
040             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) + """"
041         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.

```
042         DialogLibraries.LoadLibrary("Standard")
043         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:

```
044         oCtlList1 = oDlg.GetControl("ListBox1")
045         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.

```
046         Select Case oDlg.Execute()
047         Case 1  'Case 1 bedeutet die Betätigung des Buttons "OK"
048         Case 0  'Wenn Button "Abbrechen"
049            inWiederholung = 0
050         End Select
051      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:

```
052      Case 1
053         stInhalt1 = oCtlList1.getSelectedItem() 'Wert aus Listbox1 auslesen ...
054         REM ... und den dazugehoerigen ID-Wert bestimmen.
```

Der ID-Wert des ersten Listenfeldes wird in der Variablen **inLB1** gespeichert.

```
055         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.

```
056         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 gestar-

tet werden, die hier gleichzeitig erstellt und, durch ein Semikolon getrennt, weitergeleitet werden.

```
057         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.

```
058         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.

```
059     IF stNebentabelle = aTabelle(5) THEN
060         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.

```
061 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.

```
062     IF aFremdTabellen(i,1) = stFeldbezeichnung THEN
063         inUnique = 1
064     ELSE
065         ReDim Preserve aSpalten(inZaehler)
066         aSpalten(inZaehler) = oAbfrageergebnis.getString(1)
067         inZaehler = inZaehler + 1
068     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.

```
069 IF inUnique = 1 THEN
```

```
070     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 + "' )"
071     IF inZaehler > 0 THEN
072        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)**.

```
073        stFeldbezeichnung = ""
074        FOR ink = LBound(aSpalten()) TO UBound(aSpalten())
075           stFeldbezeichnung = stFeldbezeichnung + " AND """ + aSpalten(ink)
              + """ = ""a"".""" + aSpalten(ink) + """ "
```

Die SQL-Teilstücke werden für eine korrelierte Unterabfrage zusammengefügt.

```
076        NEXT
077        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.

```
078        stSql=stSql + stFeldbezeichnung + "GROUP BY (""" + stFeldgruppe + """) ) < 1"
079     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

```
080     ELSE
081        stSql = "UPDATE """ + aFremdTabellen(i,0) + """ SET """ + aFremdTabellen(i,1)
           + """='" + inLB2 + "' WHERE """ + aFremdTabellen(i,1) + """='" + inLB1 + "'"
082  END IF
083  oSQL_Anweisung.executeQuery(stSql)
084  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.

```
085  stSql = "DELETE FROM """ + aTabelle(5) + """ WHERE ""ID""='" + inLB1 + "'"
086  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.

```
087  ELSEIF oCtlCheck1.State = 1 THEN
088     stBedingung = ""
089     IF stNebentabelle = aTabelle(5) THEN
090        FOR i = LBound(aFremdTabellen()) TO UBound(aFremdTabellen())
091           stBedingung = stBedingung + """ID"" NOT IN (SELECT """ +
              aFremdTabellen(i,1) + """ FROM """ + aFremdTabellen(i,0) + """) AND "
092        NEXT
093     ELSE
094        FOR i = LBound(aFremdTabellen2()) TO UBound(aFremdTabellen2())
095           stBedingung = stBedingung + """ID"" NOT IN (SELECT """ +
              aFremdTabellen2(i,1) + """ FROM """ + aFremdTabellen2(i,0) + """) AND "
```

```
096         NEXT
097     END IF
```

Das letzte **AND** muss abgeschnitten werden, da sonst die Löschanweisung mit einem **AND** enden würde:

```
098     stBedingung = Left(stBedingung, Len(stBedingung) - 4)
099     stSql = "DELETE FROM """ + stNebentabelle + """ WHERE " + stBedingung + ""
100     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 *Interne Datenbanken sicher schließen* erwähnte Prozedur.

```
101     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.

```
102     oDlg.endExecute()  'Dialog beenden ...
103     oDlg.Dispose()     '... und aus dem Speicher entfernen
104 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

```
001 Sub DBOpen(Optional oEvent As Object)
002     If GlobalScope.BasicLibraries.hasByName("Access2Base") then
003         GlobalScope.BasicLibraries.loadLibrary("Access2Base")
004     End If
005     Call Application.OpenConnection(ThisDatabaseDocument)
006 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.

Python als Makrosprache für Datenbanken

Basic ist die Makrosprache, die LibreOffice standardmäßig unterstützt. Der Zugang über Basic ist deswegen am einfachsten gestaltet. Manchmal bieten andere Makrosprachen aber Bibliotheken an, die ebenfalls gerne mit LibreOffice genutzt werden. Dieses Kapitel soll nur den Einstieg dazu liefen, so dass auch Abfragen an Datenbanken über Python funktionieren.

Zum Start mit Python empfiehlt die Hilfe die Erweiterung APSO (*https://extensions.libreoffice.org/extensions/apso-alternative-script-organizer-for-python*). Diese Erweiterung wird auch im Folgenden teilweise genutzt.

Speicherort für das erste Python Makro

Makros in Python können nicht mit dem internen Editor über Module einfach erstellt und aufgerufen werden. Prinzipiell reicht zur Erstellung ein einfacher Texteditor aus. Die damit erzeugte

Datei sollte lediglich die Endung *.py erhalten. Hier ein kleines Beispiel für ein neu geöffnetes Calc-Dokument:

```
001 def HelloWorld2Calc():
002     doc = XSCRIPTCONTEXT.getDocument()
003     cell = doc.getCurrentSelection()
004     cell.setString('Hallo Welt!')
```

Das so erstellte Script wird als test.py abgespeichert. Jetzt muss der Pfad gesucht werden, der im eigenen Benutzerverzeichnis von LibreOffice liegt: **.../libreoffice/4/user/Scripts/python/**. Wird die Datei «test.py» in dem Verzeichnis **python** abgelegt, so kann sie über **Extras → Makros → Makro ausführen...** ausgeführt werden. Das Modul heißt «test» (von «test.py») und der Name des Makros, die Prozedur, ist «HelloWorld2Calc» (von der Zeile, die mit **def** beginnt).

Speicherort für Module zum Importieren in ein Makro

Um auf andere in Python erstellte Module zugreifen zu können müssen diese Module in einem Pfad liegen, den LibreOffice kennt. Andere *.py-Dateien im gleichen Verzeichnis **python** können nicht über einen **import**-Befehl in die eigene Datei importiert werden.

In dem Verzeichnis **.../libreoffice/4/user/Scripts/python/** wird ein Unterverzeichnis **pythonpath** erstellt. In dieses Verzeichnis können andere *.py-Dateien kopiert werden, auf die das eigene Makro zugreifen können soll. Dort wird dann auch automatisch beim ersten Aufruf ein Unterverzeichnis **__pycache__** erstellt, in dem die für Python ausführbaren Dateien abgelegt werden.

Verwaltung von Modulen in Bibliotheken

In Basic bilden mehrere Prozeduren («Makros») ein Modul. Mehrere Module bilden dann zusammen eine Bibliothek. Bei entsprechend vielen Modulen ist die Zusammenfassung in Bibliotheken sinnvoll. So eine Bibliothek wird einfach als zusätzliches Verzeichnis in **.../libreoffice/4/user/Scripts/python/** erstellt. Die eigentliche Python-Datei mit der Endung *.py wird dann in diesem Verzeichnis abgelegt. Damit haben dann die Makros in Python die gleiche Verwaltungsstruktur wie die bisher bekannten Makros in StarBasic.

Abfrage an eine geöffnete Datenbankdatei

```
001 def query():
002     doc = XSCRIPTCONTEXT.getDocument()
003     src = doc.CurrentController
004     if not src.isConnected():
005         src.connect()
006     if src.isConnected():
007         conn = src.ActiveConnection
008         sqlStatement = conn.createStatement()
009         sql = 'SELECT * FROM "tbl_Person"'
010         result = sqlStatement.executeQuery(sql)
011         result.next()
012         content = result.getString(1)
```

Die Prozedur wird als «query» gestartet. LibreOffice stellt über **XSCRIPTCONTEXT** einige wichtige Methoden zur Verfügung. Hier wird auf das aktuelle Dokument zugegriffen.

Bei sämtlichen Codes kommt es genau auf die Schreibweise (Groß- und Kleinschreibung) an.

Eine **if**-Schleife beginnt mit **if**, gefolgt von der Bedingung. Ist die Bedingung erfüllt, so wird alles ausgeführt, was nach dem Doppelpunkt notiert ist.

Die Verbindung zur geöffneten Datenbankdatei wird hergestellt. Wie in Basic wird eine Abfrage erstellt, das Ergebnis einer Abfrage zwischengespeichert und der erste Wert der Abfrage über **result.next()** angesteuert. Als Inhalt wird beispielhaft nur der Wert des ersten Feldes der ersten Zeile ausgelesen. Mit dem Inhalt kann jetzt weiter gearbeitet werden. Er lässt sich aber lei-

der nicht so ohne Probleme sichtbar machen, da in Python die Ausgabe auf der Konsole erfolgen würde und die Funktion **msgbox** aus Basic unbekannt ist.

Die Funktion wird hier verfügbar gemacht, indem aus der Erweiterung «APSO» durch Entpacken die Dateien «apso_utils.py» und «theconsole.py» kopiert und in das Verzeichnis **.../libreoffice/4/user/Scripts/python/pythonpath** eingefügt werden. Jetzt wird der obige Pythoncode erweitert:

```
001 from apso_utils import msgbox
```

```
001 def query():
002     doc = XSCRIPTCONTEXT.getDocument()
003     src = doc.CurrentController
004     if not src.isConnected():
005         src.connect()
006     if src.isConnected():
007         conn = src.ActiveConnection
008         sqlStatement = conn.createStatement()
009         sql = 'SELECT * FROM "tbl_Person"'
010         result = sqlStatement.executeQuery(sql)
011         result.next()
012         content = result.getString(1)
013         msgbox(content)
```

Aus der Datei «apso_utils» wird die Funktion **msgbox** importiert. Diese Funktion wird in Zeile 13 aufgerufen und zeigt den Ergebniswert an.

Die Datei «apso_utils» bietet noch mehr Funktionen wie das erleichterte Einbinden von xray oder mri, zwei Tools, die zur Untersuchung der zur Verfügung stehenden Objekte und Befehle genutzt werden können. Deswegen schien es mir sinnvoll, diese Datei für den weiteren Zugriff bereit zu stellen. Die Datei «apso_utils.py» benötigt die Datei «theconsole.py» für einige Funktionen. Deswegen musste auch diese Datei in das entsprechende Verzeichnis importiert werden.

Abfrage an eine registrierte Datenbank

In der registrierten Datenbank des folgenden Beispiels sind in einer Tabelle Bilder gespeichert, die ausgelesen werden sollen. Die folgende Prozedur liest die Bilder aus der Firebird-Datenbank aus und schreibst sie mit dem entsprechend angegebenen Namen in das temporäre Verzeichnis des genutzten Linux-Systems. Diese Prozedur stellt Methoden zum Auslesen zur Verfügung, die ich in Basic so nicht umgesetzt bekam. Dort konnte ich Bilder, die deutlich größer als ein Icon waren, nicht problemlos aus Firebird exportieren.

```
001 import io
002 import uno
```

```
001 def ConnectRegisteredDB():
002     ctx = uno.getComponentContext()
003     smgr = ctx.getServiceManager()
004     obj = smgr.createInstanceWithContext('com.sun.star.sdb.DatabaseContext', ctx)
005     db = obj['Beispiel_Druck_Writer_Tabellen_FB']
006     conn = db.getConnection('', '')
007     sql = 'SELECT "Name", "Bild" FROM "tbl_Image"'
008     sqlStatement = conn.createStatement()
009     result = sqlStatement.executeQuery(sql)
010     while result.next():
011         name = result.getString(1)
012         stream = result.getBlob(2)
013         size, data = stream.readBytes(io.BytesIO(), stream.available())
014         myfile = open('/tmp/bild'+name, 'wb')
015         myfile.write(data.value)
016         conn.close()
```

Zuerst werden Module eingebunden, die das Makro nutzen will. Die beiden zu importierenden Module stellt LibreOffice direkt zur Verfügung.

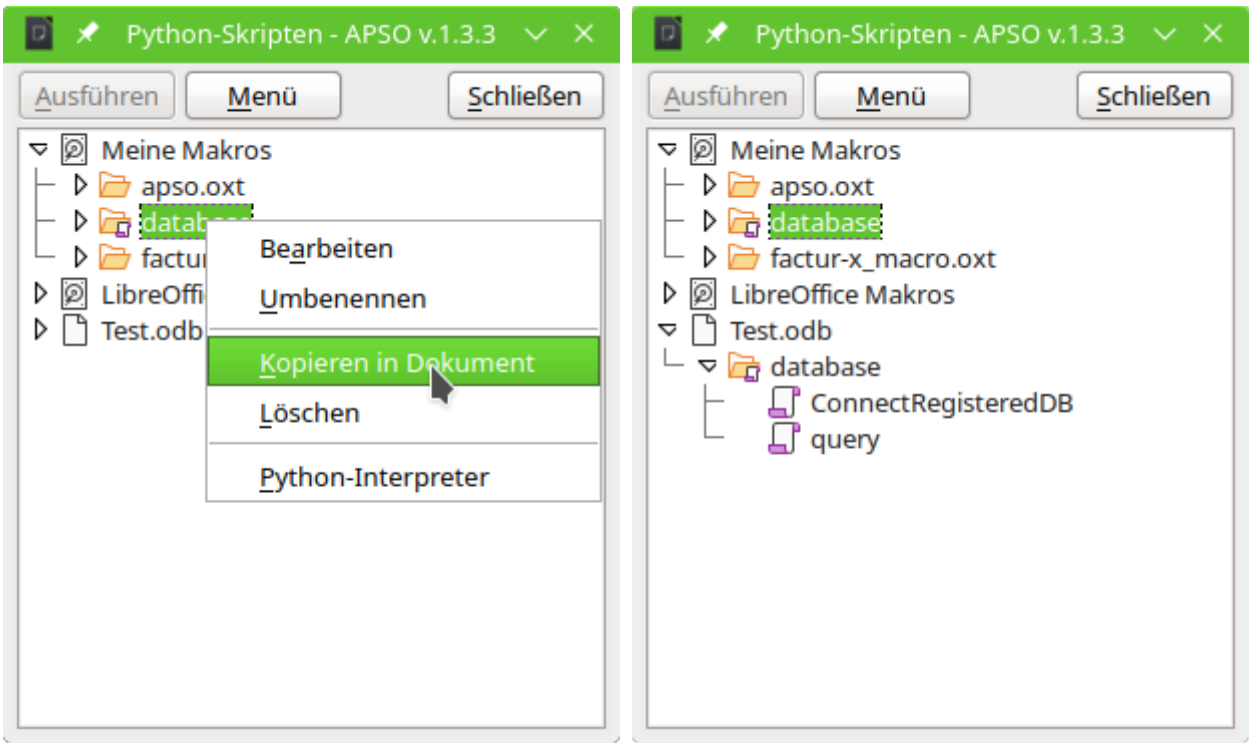
Die Zeilen 2 bis 4 sind direkt aus der Hilfe von LibreOffice entnommen. Sie entsprechen zusammen dem Befehl **CreateUnoService** aus Basic. Damit wird dann auf die registrierten Datenbanken zugegriffen. Die hier genutzte registrierte Datenbank heißt «Beispiel_Druck_Writer_Tabellen_FB» und ist zu dem Zeitpunkt nicht geöffnet. Aus der Datenbank wird ein Textfeld und ein Feld des Typs BLOB (für Bilder) ausgelesen. Dies geschieht für alle Datensätze mit einer **while**-Schleife (Zeile 10). Nur der Doppelpunkt kennzeichnet den Start der Schleife.

In Zeile 13 wird der Datenstrom ausgelesen. Diese Art des Auslesens ist mir unter Basic nicht geglückt.

Zeile 14 dient dazu eine Datei zum Schreiben mit binärem Inhalt **'wb'** zu öffnen. In Zeile 15 wird dann der ausgelesene Datenstrom geschrieben. Zum Schluss wird dann die Verbindung zur Datenbank aufgehoben.

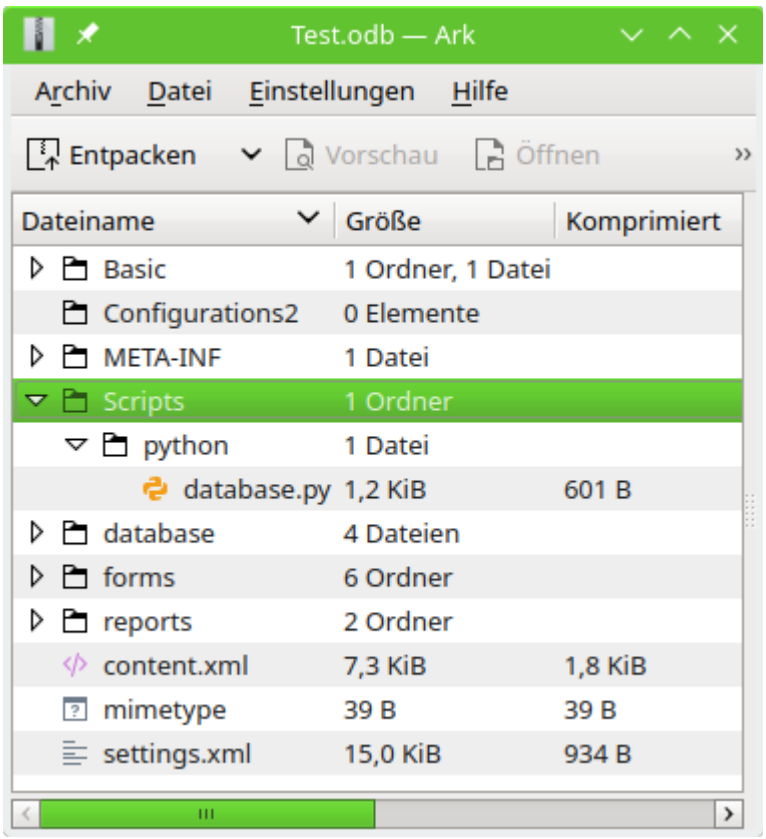
Fertige Module in die Base-Datei einbinden

Viele Makros für Base sind spezifisch an eine Datenbank angepasst. Dies liegt schon allein daran, dass Abfragen an unterschiedliche Datenquellen gestellt werden, Formulare unterschiedlich konstruiert sind usw. Auch für die Weitergabe von Base-Dateien ist es wichtig, dass die Funktionen erhalten bleiben. Deswegen müssen die fertigen Bibliotheken irgendwie in die Base-Datei eingebunden werden. Dies geschieht am einfachsten mit der Extension APSO:



Über **Extras → Makros → Python-Scripte verwalten** wird der Dialog von APSO gestartet. Das Modul aus dem Ordner **Meine Makros** wird ausgesucht. Mit einem rechten Mausklick auf das Modul (nicht auf eine der Prozeduren in dem Modul) wird das gesamte Modul in das geöffnete Datenbankdokument (hier: «Test.odb») kopiert. Die Bibliothek kann auch in der Datenbankdatei weiter bearbeitet, wieder exportiert oder auch gelöscht werden. Das Bearbeiten gestaltet sich hier natürlich etwas schwieriger, da die *.py-Dateien nicht mehr direkt über die Verzeichnisstruktur des Betriebssystems erreichbar sind sondern in der gepackten Base-Datei liegen.

Der Import in die Datenbankdatei kann natürlich auch über die Bordmittel des Betriebssystems erfolgen. Dafür reicht ein Packprogramm aus, mit dem die Datenbankdatei geöffnet werden kann.



In der *.odb_Datei muss für die Pythonscripte ein Verzeichnis «Scripts» und ein Unterverzeichnis «python» erstellt werden. In das Unterverzeichnis «python» werden dann alle benötigten Module geladen.

Vorsicht

Mit APSO wird nur das aktuelle Modul in die Datenbankdatei kopiert. Werden aber für einzelne Prozeduren Module von außen importiert, wie dies in der *Abfrage an eine geöffnete Datenbankdatei* der Fall ist, dann fehlen diese Module auf anderen Systemen. Dort sind nur die Module vorhanden, die standardmäßig mit LibreOffice ausgeliefert wurden, nicht aber die in dem Beispiel genutzten Bibliotheken aus APSO. Die Funktion «msgbox» steht also anderen Nutzern nicht zur Verfügung. Hierfür müsste auch das Verzeichnis «pythonpath» mit den darin enthaltenen Dateien in die *.odb-Datei kopiert werden.