

LibreOffice
The Document Foundation

Base

Kapitel 9

Makros

Copyright

Dieses Dokument unterliegt dem Copyright © 2012. Die Beitragenden sind unten aufgeführt. Sie dürfen dieses Dokument unter den Bedingungen der GNU General Public License (<http://www.gnu.org/licenses/gpl.html>), Version 3 oder höher, oder der Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0/>), Version 3.0 oder höher, verändern und/oder weitergeben.

Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.

Fast alle Hardware- und Softwarebezeichnungen und weitere Stichworte und sonstige Angaben, die in diesem Buch verwendet werden, sind als eingetragene Marken geschützt.

Da es nicht möglich ist, in allen Fällen zeitnah zu ermitteln, ob ein Markenschutz besteht, wird das Symbol (R) in diesem Buch nicht verwendet.

Mitwirkende/Autoren

Robert Großkopf

Jost Lange

Jochen Schiffers

Jürgen Thomas

Michael Niedermaier

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 1.3.2013. Basierend auf der LibreOffice Version 4.0

Anmerkung für Macintosh Nutzer

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

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

Inhalt

Allgemeines zu Makros	4
Makros in Base	5
Makros benutzen	5
Makros zuweisen	6
Ereignisse eines Formulars	6
Ereignisse innerhalb eines Formulars	7
Bestandteile von Makros	8
Der «Rahmen» eines Makros	8
Variablen definieren	8
Zugriff auf das Formular	9
Zugriff auf Elemente eines Formulars	9
Zugriff auf die Datenbank	10
Datensätze lesen und benutzen	11
Datensätze bearbeiten – neu anlegen, ändern, löschen	13
Kontrollfelder prüfen und ändern	14
Englische Bezeichner in Makros	15
Eigenschaften bei Formularen und Kontrollfeldern	15
Methoden bei Formularen und Kontrollfeldern	21
Bedienbarkeit verbessern	25
Automatisches Aktualisieren von Formularen	25
Filtern von Datensätzen	26
Daten aus Textfeldern auf SQL-Tauglichkeit vorbereiten	29
Suchen von Datensätzen	30
Kombinationsfelder als Listenfelder mit Eingabemöglichkeit	32
Textanzeige im Kombinationsfeld	32
Übertragen eines Fremdschlüsselwertes vom Kombinationsfeld zum numerischen Feld ...	35
Kontrollfunktion für die Zeichenlänge der Kombinationsfelder	40
Aufruf der Prozedur zum Anzeigen des Textes	41
Aufruf der Prozedur zur Textspeicherung	41
Navigation von einem Formular zum anderen	42
Datenbankaufgaben mit Makros erweitert	43
Verbindung mit Datenbanken erzeugen	43
Datenbanksicherungen erstellen	44
Datenbanken komprimieren	45
Tabellenindex heruntersetzen bei Autowert-Feldern	45
Serienbriefdruck aus Base heraus	46
Aufruf von Anwendungen zum Öffnen von Dateien	47
Aufruf eines Mailprogramms mit Inhaltsvorgaben	48
Mauszeiger beim Überfahren eines Links ändern	49
Formulare ohne Symbolleisten präsentieren	49
Formulare ohne Symbolleisten in einem Fenster	49
Formulare im Vollbildmodus	50
Dialoge	51

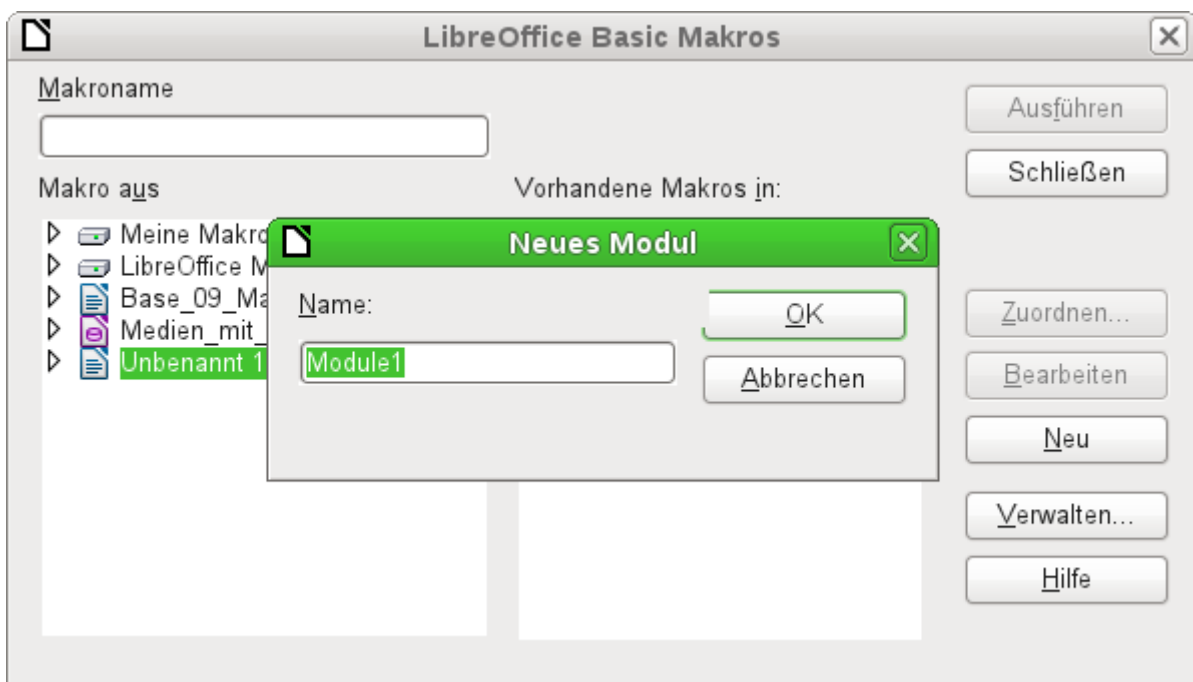
Allgemeines zu Makros

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

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

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

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



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

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

```
REM ***** BASIC *****  
  
Sub Main  
  
End Sub
```

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

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

Einige Grundprinzipien zur Nutzung des Basic-Codes in LibreOffice:

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

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

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

Hinweis

Makros in diesem Kapitel sind entsprechend den Vorgaben aus dem Makro-Editor von LibreOffice eingefärbt:

Makro-Bezeichner
 Makro-Kommentar
 Makro-Operator
 Makro-Reservierter-Ausdruck
 Makro-Zahl
 Makro-Zeichenkette

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

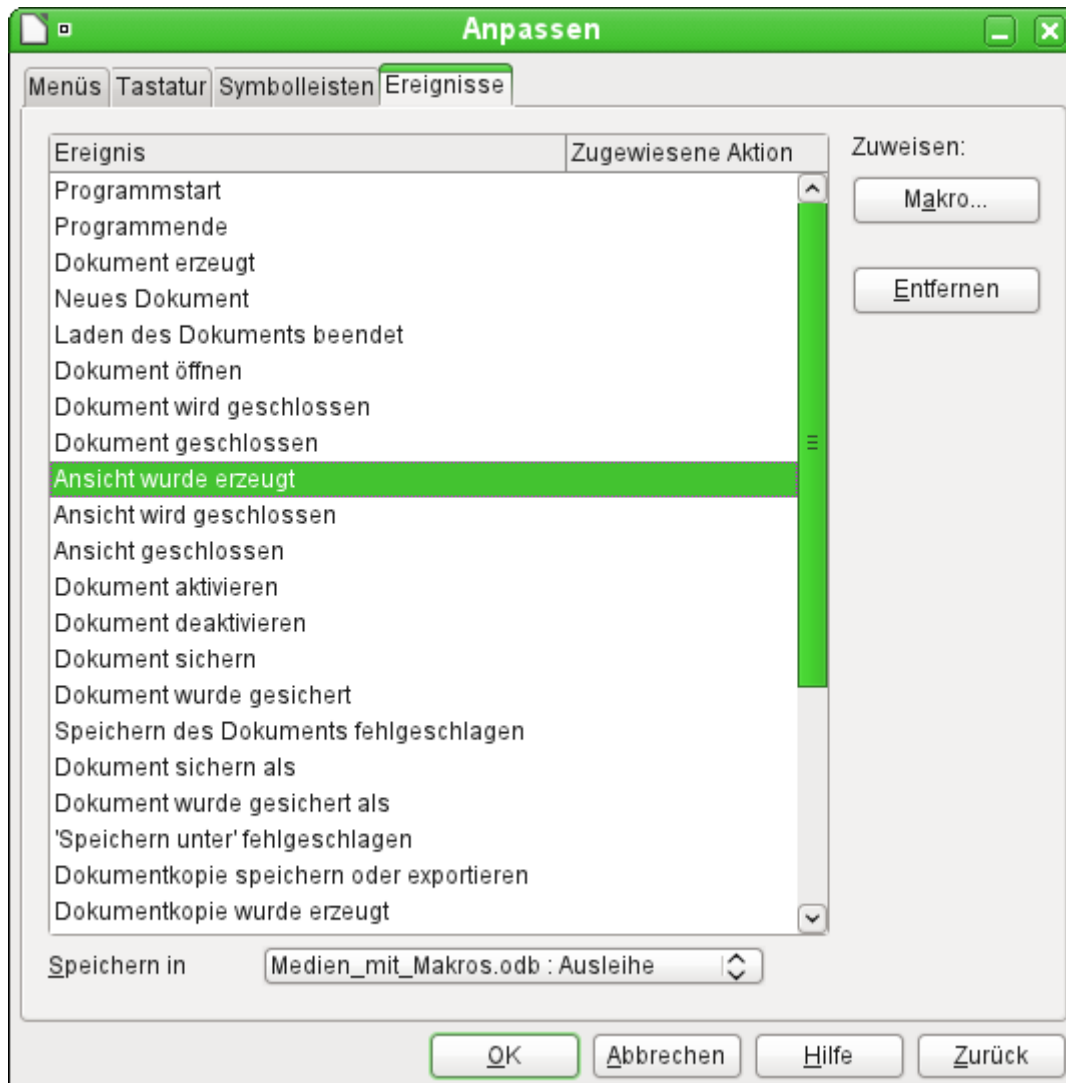
Der «direkte Weg» ist vor allem dann nicht möglich – auch nicht zu Testzwecken –, wenn eines der Objekte **thisComponent** (siehe den Abschnitt «[Zugriff auf das Formular](#)») oder **oEvent** (siehe den Abschnitt «[Zugriff auf Elemente eines Formulars](#)») benutzt wird.

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

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

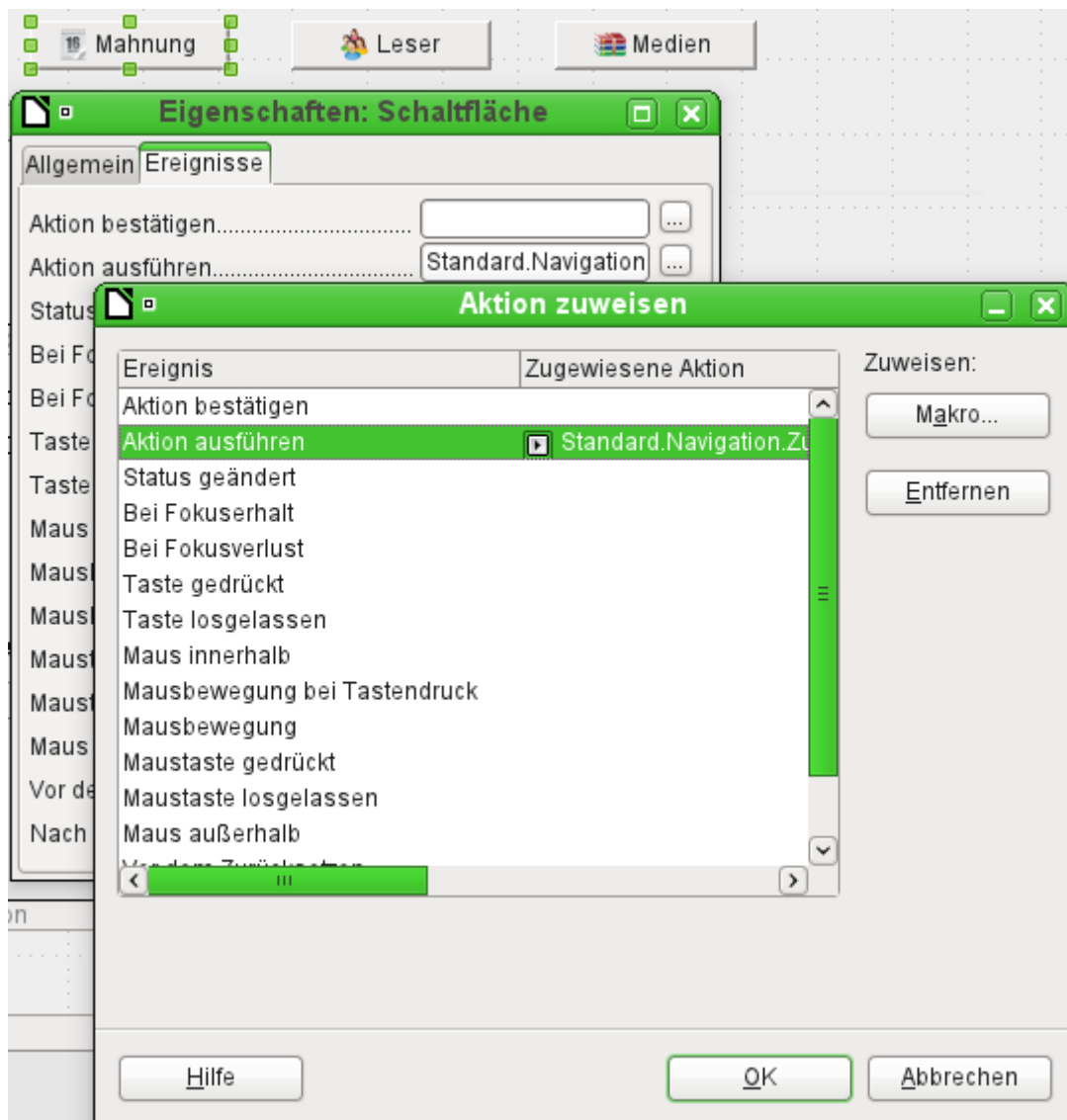


- Rufen Sie im Formularentwurf über **Extras** → **Anpassen** das Register **Ereignisse** auf.
- Wählen Sie das passende Ereignis aus der Gruppe «Ansicht» aus (die anderen kommen nicht infrage).
- Suchen Sie über die Schaltfläche **Makro** das dafür definierte Makro und bestätigen Sie diese Auswahl.
- Unter **Speichern in** ist das Formular anzugeben.

Dann kann diese Zuweisung mit **OK** bestätigt werden.

Ereignisse innerhalb eines Formulars

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



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

Über mehrfaches **OK** wird diese Zuweisung bestätigt.

Bestandteile von Makros

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

Der «Rahmen» eines Makros

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

```
SUB Ausleihe_aktualisieren
END SUB

SUB Zu_Formular_von_Formular(oEvent AS OBJECT)
END SUB

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

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

Variablen definieren

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

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

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

Üblich ist es, durch den ersten Buchstaben den Datentyp deutlich zu machen. 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 StarBasic 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.

Zugriff auf das Formular

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

```
oDoc = thisComponent
oDrawpage = oDoc.drawpage
oForm = oDrawpage.forms.getByName("Filter")
```

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

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

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

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

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

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

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

Zugriff auf Elemente eines Formulars

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

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

Dieser Weg funktioniert immer dann, wenn bekannt ist, mit welchem Element das Makro arbeiten soll. Wenn aber im ersten Schritt zu prüfen ist, welches Ereignis das Makro gestartet hat, ist der o.g. Weg über **oEvent** sinnvoll. Dann wird die Variable innerhalb des Makro-"Rahmens" deklariert und beim Start des Makros zugewiesen. Die Eigenschaft **source** liefert immer dasjenige Element, das das Makro gestartet hat; die Eigenschaft **model** beschreibt das Kontrollfeld im Einzelnen:

```
SUB Auswahl_bestatigen(oEvent AS OBJECT)
  DIM btnOK AS OBJECT
  btnOK = oEvent.source.model
END
```

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

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

Zugriff auf die Datenbank

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

Die Verbindung zur Datenbank

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

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

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

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

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

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

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

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

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

Ergänzende Hinweise zur Datenbankverbindung stehen im Abschnitt «[Verbindung mit Datenbanken erzeugen](#)».

SQL-Befehle

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

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

Um *Daten abzufragen*, wird mit dem Befehl die Methode **executeQuery** aufgerufen und ausgeführt; das Ergebnis wird anschließend ausgewertet:

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

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

```
stSql = "DROP TABLE ""Suchtmp"" IF EXISTS"
```

```
iResult = oSQL_Statement.executeUpdate(stSql)
```

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

Vorbereitete SQL-Befehle mit Parametern

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

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

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

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

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

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

Die vollständige Liste der Zuweisungen findet sich im Abschnitt «[Parameter für vorbereitete SQL-Befehle](#)».

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

- [SQL-Injection \(Wikipedia\)](#)
- [Why use PreparedStatement \(Java JDBC\)](#)
- [SQL-Befehle \(Einführung in SQL\)](#)

Datensätze lesen und benutzen

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

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

Mithilfe des Formulars

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

```
DIM ID AS LONG
```

```

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

```

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

Hinweis

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

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

```

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

```

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

- Es gibt keine Methode für Daten des Typs **Decimal**, **Currency** o.ä., also für kaufmännisch exakte Berechnungen. Da Basic automatisch die passende Konvertierung vornimmt, kann ersatzweise **getDouble** verwendet werden.
- Bei **getBoolean** ist zu beachten, wie in der Datenbank «Wahr» und «Falsch» definiert sind. Die «üblichen» Definitionen (logische Werte, 1 als «Wahr») werden richtig verarbeitet.
- Datumsangaben können nicht nur mit dem Datentyp **DATE** definiert werden, sondern auch (wie oben) als **util.Date**. Das erleichtert u.a. Lesen und Ändern von Jahr, Monat, Tag.
- Bei ganzen Zahlen sind Unterschiede der Datentypen zu beachten. Im obigen Beispiel wird **getLong** verwendet; auch die Basic-Variable ID muss den Datentyp **Long** erhalten, da dieser vom Umfang her mit **Integer** aus der Datenbank übereinstimmt.

Die vollständige Liste dieser Methoden findet sich im Abschnitt «[Datenzeilen bearbeiten](#)».

Ergebnis einer Abfrage

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

```

WHILE oResult.next           ' einen Datensatz nach dem anderen verarbeiten
  rem übernimm die benötigten Werte in einzelne Variable
  sVar = oResult.getString(1)
  rem mach etwas mit diesen Werten
WEND

```

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

Die Anzahl der Zeilen, die die Ergebnismenge enthält, kann so bestimmt werden:

```

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

```

Mithilfe eines Kontrollfelds

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

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

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

In einer Datenmenge navigieren

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

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

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

Datensätze bearbeiten – neu anlegen, ändern, löschen

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

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

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

Inhalt eines Kontrollfelds ändern

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

```
oDateField.BoundField.updateDate( CDateToIso(Now) )
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.

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:

```
oForm.updateDate(3, CdateToIso(Now) )
oForm.updateString(4, "ein Text")
oForm.updateDouble(6, 3.14)
oForm.updateInt(7, 16)
oForm.updateRow()
```

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

Zeilen anlegen, ändern, löschen

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

1. Wähle den aktuellen Datensatz.
2. Ändere die gewünschten Werte, wie im vorigen Abschnitt beschrieben.
3. Bestätige die Änderungen mit folgendem Befehl:
`oForm.updateRow()`
4. Als Sonderfall ist es auch möglich, die Änderungen zu verwerfen und den vorherigen Zustand wiederherzustellen:
`oForm.cancelRowUpdates()`

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

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

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

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

Kontrollfelder prüfen und ändern

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

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

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

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

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

Englische Bezeichner in Makros

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

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

Die Übersichten nennen folgende Angaben:

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

Weitere Informationen finden sich vor allem in der [API-Referenz](#) mit Suche nach der englischen Bezeichnung des Kontrollfelds.

Eigenschaften bei Formularen und Kontrollfeldern

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

Schrift

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

Name	Datentyp	L/S	Eigenschaft
FontName	string	L+S	Schriftart.
FontHeight	single	L+S	Schriftgröße.
FontWeight	single	L+S	Schriftstärke.
FontSlant	integer	L+S	Art der Schrägstellung.

Name	Datentyp	L/S	Eigenschaft
FontUnderline	integer	L+S	Art der Unterstreichung.
FontStrikeout	integer	L+S	Art des Durchstreichens.

Formular

Englische Bezeichnung: *Form*

Name	Datentyp	L/S	Eigenschaft
ApplyFilter	boolean	L+S	Filter aktiviert.
Filter	string	L+S	Aktueller Filter für die Datensätze.
FetchSize	long	L+S	Anzahl der Datensätze, die «am Stück» geladen werden.
Row	long	L	Nummer der aktuellen Zeile.
RowCount	long	L	Anzahl der Datensätze.

Einheitlich für alle Arten von Kontrollfeld

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

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

Einheitlich für viele Arten von Kontrollfeld

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

Textfeld – weitere Angaben

Englische Bezeichnung: *TextField*

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

Numerisches Feld

Englische Bezeichnung: *NumericField*

Name	Datentyp	L/S	Eigenschaft
ValueMin	double	L+S	Minimalwert zur Eingabe.
ValueMax	double	L+S	Maximalwert zur Eingabe.
Value	double	L+(S)	Aktueller Wert <i>nicht für Werte aus der Datenmenge verwenden.</i>
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.

Datumfeld

Englische Bezeichnung: *DateField*

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

Name	Datentyp	L/S	Eigenschaft
DateMin	long	L+S	Minimalwert zur Eingabe.
DateMax	long	L+S	Maximalwert zur Eingabe.
Date	long	L+(S)	Aktueller Wert <i>nicht für Werte aus der Datenmenge verwenden.</i>
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.
DefaultDate	long	L+S	Standardwert.
DropDown	boolean	L+S	Aufklappbaren Monatskalender anzeigen.

Zeitfeld

Englische Bezeichnung: *TimeField*

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

Name	Datentyp	L/S	Eigenschaft
TimeMin	long	L+S	Minimalwert zur Eingabe.
TimeMax	long	L+S	Maximalwert zur Eingabe.
Time	long	L+(S)	Aktueller Wert <i>nicht für Werte aus der Datenmenge verwenden.</i>
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	L+S	Standardwert.

Währungsfeld

Englische Bezeichnung: *CurrencyField*

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

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

Formatiertes Feld

Englische Bezeichnung: *FormattedControl*

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

Name	Datentyp	L/S	Eigenschaft
CurrentValue	variant	L	Aktueller Wert des Inhalts; der konkrete Datentyp hängt vom Inhalt des Feldes und dem Format ab.
EffectiveValue		L+(S)	
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. <i>Es gibt kein einfaches Verfahren, das Format durch ein Makro zu ändern.</i>
EnforceFormat	boolean	L+S	Formatüberprüfung: Bereits während der Eingabe sind nur zulässige Zeichen und Kombinationen möglich.

Listenfeld

Englische Bezeichnung: *ListBox*

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

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

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

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

Kombinationsfeld

Englische Bezeichnung: *ComboBox*

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

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

Name	Datentyp	L/S	Eigenschaft
Autocomplete	boolean	L+S	Automatisch füllen.
StringItemList	array of string	L+S	Listeneinträge, die zur Auswahl zur Verfügung stehen.
ItemCount	integer	L	Anzahl der vorhandenen Listeneinträge.
DropDown	boolean	L+S	Aufklappbar.
LineCount	integer	L+S	Anzahl der angezeigten Zeilen im aufgeklappten Zustand.
Text	string	L+S	Aktuell angezeigter Text.
DefaultText	string	L+S	Standardeintrag.
ListSource	string	L+S	Name der Datenquelle, die die Listeneinträge liefert.

Name	Datentyp	L/S	Eigenschaft
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).

Maskiertes Feld

Englische Bezeichnung: *PatternField*

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

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

Tabellenkontrollfeld

Englische Bezeichnung: *GridControl*

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

Beschriftungsfeld

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

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

Gruppierungsrahmen

Englische Bezeichnung: *GroupBox*

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

Schaltfläche

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

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

Navigationsleiste

Englische Bezeichnung: *NavigationBar*

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

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

Methoden bei Formularen und Kontrollfeldern

Die Datentypen der Parameter werden durch Kürzel angedeutet:

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

In einer Datenmenge navigieren

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

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

Name	Datentyp	Beschreibung
Prüfungen für die Position des Cursors		
isBeforeFirst	boolean	Der Cursor steht vor der ersten Zeile, wenn der Cursor nach dem Einlesen noch nicht gesetzt wurde.
isFirst	boolean	Gibt an, ob der Cursor auf der ersten Zeile steht.
isLast	boolean	Gibt an, ob der Cursor auf der letzten Zeile steht.
isAfterLast	boolean	Der Cursor steht hinter der letzten Zeile, wenn er von der letzten Zeile aus mit <i>next</i> weiter gesetzt wurde.
getRow	long	Nummer der aktuellen Zeile
Setzen des Cursors		

Name	Datentyp	Beschreibung
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 ursprünglichen Werte der aktuellen Zeile neu ein.
rowInserted	boolean	Gibt an, ob es sich um eine neue Zeile handelt.
rowUpdated	boolean	Gibt an, ob die aktuelle Zeile geändert wurde.
rowDeleted	boolean	Gibt an, ob die aktuelle Zeile gelöscht wurde.

Datenzeilen bearbeiten

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

Name	Datentyp	Beschreibung
Maßnahmen für die ganze Zeile		
insertRow	–	Speichert eine neue Zeile.
updateRow	–	Bestätigt Änderungen der aktuellen Zeile.
deleteRow	–	Löscht die aktuelle Zeile.
cancelRowUpdates	–	Macht Änderungen der aktuellen Zeile rückgängig.
moveToInsertRow	–	Wechselt den Cursor in die Zeile für einen neuen Datensatz.
moveToCurrentRow	–	Kehrt nach der Eingabe eines neuen Datensatzes zurück zur vorherigen Zeile.
Werte lesen		
getString(c)	string	Liefert den Inhalt der Spalte als Zeichenkette.
getBoolean(c)	boolean	Liefert den Inhalt der Spalte als Wahrheitswert.
getByte(c)	byte	Liefert den Inhalt der Spalte als einzelnes Byte.
getShort(c)	short	Liefert den Inhalt der Spalte als ganze Zahl.
getInt(c)	integer	
getLong(c)	long	

Name	Datentyp	Beschreibung
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.		
wasNull	boolean	Gibt an, ob der Wert der zuletzt gelesenen Spalte NULL war.
Werte speichern		
updateNull(c)	–	Setzt den Inhalt der Spalte c auf NULL.
updateBoolean(c,b)	–	Setzt den Inhalt der Spalte c auf den Wahrheitswert b.
updateByte(c,x)	–	Speichert in Spalte c das angegebene Byte x.
updateShort(c,n)	–	Speichert in Spalte c die angegebene ganze Zahl n.
updateInt(c,n)	–	
updateLong(c,n)	–	
updateFloat(c,n)	–	
updateDouble(c,n)	–	Speichert in Spalte c die angegebene Dezimalzahl n.
updateString(c,s)	–	Speichert in Spalte die angegebene Zeichenkette s.
updateBytes(c,x)	–	Speichert in Spalte das angegebene Byte-Array x.
updateDate(c,d)	–	Speichert in Spalte das angegebene Datum d.
updateTime(c,d)	–	Speichert in Spalte den angegebenen Zeitwert d.
updateTimestamp(c,d)	–	Speichert in Spalte den angegebenen Zeitstempel d.

Einzelne Werte bearbeiten

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

Hinweis: Damit eine Änderung in die Datenbank übernommen wird, ist sie durch «updateRow» bzw. «insertRow» ausdrücklich zu bestätigen.

Name	Datentyp	Beschreibung
Werte lesen		
getString	string	Liefert den Inhalt der Spalte als Zeichenkette.

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

Parameter für vorbereitete SQL-Befehle

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

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

Bedienbarkeit verbessern

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

Automatisches Aktualisieren von Formularen

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

`SUB Aktualisieren`

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

Das Makro hat jetzt den Namen «*Aktualisieren*». Um sicher zu gehen, dass keine Variablen von außen eingeschleust werden gehen viele Programmierer so weit, dass sie Basic über '**Option Explicit**' gleich zu Beginn mitteilen: Erzeuge nicht automatisch irgendwelche Variablen sondern nutze nur die, die ich auch vorher definiert habe.

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

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

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

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

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

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

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

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

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

Filtern von Datensätzen

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

Siehe zu diesem Abschnitt auch die **Beispieldatenbank** «Suchen_und_Filtern.odt»

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

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

```
oForm1 = oDrawpage.forms.getByName("Filter")
oForm2 = oDrawpage.forms.getByName("Anzeige")
oFeldList1 = oForm1.getByName("Liste_1")
oFeldList2 = oForm1.getByName("Liste_2")
```

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

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

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

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

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

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

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

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

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

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

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

```
SELECT  
IFNULL( "Feld_1" || ' - ' || "Anzahl", 'leer - ' || "Anzahl" ) AS  
"Anzeige",  
"Feld_1"  
FROM
```

```

( SELECT COUNT( "ID" ) AS "Anzahl", "Feld_1" FROM "Tabelle" WHERE "ID"
IN
( SELECT "Tabelle"."ID" FROM "Filter", "Tabelle" WHERE
"Tabelle"."Feld_2" = IFNULL( "Filter"."Filter_2", "Tabelle"."Feld_2" )
)
GROUP BY "Feld_1" )
ORDER BY "Feld_1"

```

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

Die Abfrage im Einzelnen:

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

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

```

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

```

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

```

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

```

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

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

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

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

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

```

SUB Filter_Zusatzinfo(oEvent AS OBJECT)
DIM oDoc AS OBJECT
DIM oDrawpage AS OBJECT
DIM oForm1 AS OBJECT
DIM oForm2 AS OBJECT
DIM oFeldList1 AS OBJECT
DIM oFeldList2 AS OBJECT
DIM sTag AS String
sTag = 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 von dem Listenfeld, das mit der Aktion (Event) verbunden ist.

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

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

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

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

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

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

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

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

Daten aus Textfeldern auf SQL-Tauglichkeit vorbereiten

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

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

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

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

Die Funktion übergibt ihr Ergebnis durch den folgenden Aufruf:

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

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

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

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

Suchen von Datensätzen

Ebenfalls ohne Makro funktioniert auch das Suchen von Datensätzen. 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 der 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 ein 'VIEW' zu erstellen, auf den das Makro auch zugreifen kann.

Siehe zu diesem Abschnitt auch die **Beispieldatenbank** «Suchen_und_Filtern.odt»

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

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

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

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

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

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

Hinweis

SQL-Formulierungen müssen in Makros wie normale Zeichenketten zuerst einmal in doppelten Anführungsstrichen gesetzt werden. Feldbezeichnungen und Tabellenbezeichnungen stehen innerhalb der SQL-Formulierungen 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 wird, wird zuerst auf 0 gesetzt. Dann wird begonnen die Abfrage auszulesen. Da die Größe des Arrays unbekannt ist, muss immer wieder nachjustiert werden. Deshalb beginnt die Schleife damit, über '**ReDim Preserve arInhalt(inI)**' die Größe des Arrays festzulegen und den vorherigen Inhalt dabei zu sichern. Anschließend werden die Felder ausgelesen und der Zähler des Arrays um 1 heraufgesetzt. Damit kann dann das Array neu dimensioniert werden und wieder ein weiterer Wert abgespeichert werden.

```
InI = 0
IF NOT ISNULL(oAbfrageergebnis) THEN
    WHILE oAbfrageergebnis.next
        ReDim Preserve arInhalt(inI)
        arInhalt(inI) = oAbfrageergebnis.getString(1)
        inI = inI + 1
    WEND
END IF
stSql = "DROP TABLE ""Suchtmp"" IF EXISTS"
oSQL_Anweisung.executeUpdate (stSql)
```

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

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

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

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

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

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

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

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

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

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

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

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

Kombinationsfelder als Listenfelder mit Eingabemöglichkeit

Aus Kombinationsfeldern und unsichtbaren numerischen Feldern kann direkt eine Tabelle mit einem Datensatz versehen werden und der entsprechende Primärschlüssel in eine andere Tabelle eingetragen werden.

In den Vorversionen von LO oder OOo war es notwendig, die numerischen Felder per Makro unsichtbar zu machen. Dies ist unter LibreOffice nicht mehr notwendig, da die Eigenschaft 'Sichtbar' in der GUI enthalten ist.

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 Schlüsselwertwerte in separaten numerischen Feldern abgelegt. Diese separaten Felder mussten vor OOo 3.3 als «unsichtbar» geschaltet werden, da die Funktion über das Formulardesign nicht erreichbar war. Diese Funktion konnte in LibreOffice und OOo 3.3 entfernt werden. Felder können jetzt als unsichtbar deklariert werden. Die Schlüssel aus diesen Feldern 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 Feld übertragen.

Das Original dieses Moduls befindet sich in der Beispieldatenbank, die durch Makros erweitert ist.

Textanzeige im Kombinationsfeld

Diese Prozedur soll Text in den Kombinationsfeldern nach den Werten der (unsichtbaren) Fremdschlüssel-Felder aus dem Hauptformular einstellen. Dabei werden gegebenenfalls auch Listenfelder berücksichtigt, die sich auf 2 unterschiedliche Tabellen beziehen. Dies kann z.B. dann sein,

wenn bei einer Ortsangabe die Postleitzahl vom Ort abgetrennt wurde. Dann wird die Postleitzahl aus einer Tabelle ausgelesen, in der auch ein Fremdschlüssel für den Ort liegt. Im Listenfeld werden Postleitzahl und Ort zusammen angezeigt.

```
SUB TextAnzeigen(NameFormular AS STRING, NameUnterformular AS STRING,
NameSubUnterformular AS STRING, NameFeld AS STRING, NameIDFeld AS STRING,
NameTabellenFeld1 AS STRING, NameTabellenFeld2 AS STRING, Feldtrenner AS STRING,
NameTabelle1 AS STRING, OPTIONAL NameTabelle2 AS STRING, OPTIONAL NameTab12ID AS
STRING, OPTIONAL Position AS INTEGER )
```

Dieses Makro sollte an die folgenden Ereignisse des Formulars gebunden werden: 'Beim Laden'
'Nach dem Datensatzwechsel'

Die folgenden Parameter sollen optional sein. Dann müssen sie beim Aufruf der Prozedur nicht unbedingt angegeben werden. Damit kein Laufzeitfehler entsteht müssen sie vorbelegt sein.

```
IF isMissing(NameTabelle2) THEN NameTabelle2 = ""
IF isMissing(NameTab12ID) THEN NameTab12ID = ""
IF isMissing(Position) THEN Position = 2
```

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

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

```
DIM oForm AS OBJECT
DIM oSubForm AS OBJECT
DIM oSubSubForm AS OBJECT
DIM oFeld AS OBJECT
DIM oFeldList AS OBJECT
DIM stFeldwert AS STRING
DIM inID AS INTEGER
DIM oCtlView AS OBJECT
oDoc = thisComponent
oDrawpage = oDoc.Drawpage
oForm = oDrawpage.forms.getByName(NameFormular)
```

Die Lage des Feldes in dem entsprechenden Formular wird aufgesucht. Alle Formulare liegen auf der '**Drawpage**' des aktuellen Dokumentes '**thisComponent**'. Diese Prozedur deckt mit den Aufrufparametern die Möglichkeit ab, dass das Feld in einem Unterformular eines Unterformulars liegt, also 2 Ebenen unterhalb des eigentlichen Formulars. Das Feld, das den Fremdschlüssel enthält, wird anschließend als '**oFeld**' bezeichnet. Das Kombinationsfeld, was jetzt statt eines Listenfeldes existiert, wird anschließend als '**oFeldList**' bezeichnet.

```
IF NameUnterformular <> "" THEN
oSubForm = oForm.getByName(NameUnterformular)
IF NameSubUnterformular <> "" THEN
oSubSubForm = oSubForm.getByName(NameSubUnterformular)
oFeld = oSubSubForm.getByName(NameIDFeld)
oFeldList = oSubSubForm.getByName(NameFeld)
ELSE
oFeld = oSubForm.getByName(NameIDFeld)
oFeldList = oSubForm.getByName(NameFeld)
END IF
ELSE
oFeld = oForm.getByName(NameIDFeld)
oFeldList = oForm.getByName(NameFeld)
END IF
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.

Anschließend wird der Wert des Fremdschlüssel ausgelesen. Nur wenn hier ein Wert eingetragen ist, wird eine Verbindung mit der Datenquelle hergestellt.

```

IF NOT IsEmpty(oFeld.getCurrentValue()) THEN
    inID = oFeld.getCurrentValue()
    oDatenquelle = ThisComponent.Parent.CurrentController
    IF NOT (oDatenquelle.isConnected()) Then
        oDatenquelle.connect()
    End IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()

```

Die SQL-Anweisung wird nach den in dem Kombinationsfeld dargestellten Feldern formuliert. Dabei müssen verschiedene Kombination durchgetestet werden. Die weitestgehende ist, dass das Kombinationsfeld mit einer Abfrage versorgt werden muss, die auf 2 Tabellenfeldern aus unterschiedlichen Tabellen beruht. Für mehr Möglichkeiten ist diese Prozedur nicht ausgelegt. Mit der weitestgehenden Möglichkeit beginnt der Test.

```
IF NameTabellenFeld2 <> "" THEN
```

Wenn ein zweites Tabellenfeld existiert

```
IF NameTabelle2 <> "" THEN
```

... und wenn eine zweite Tabelle existiert wird der folgende SQL-Code erstellt:

```

IF Position = 2 THEN
    stSql = "SELECT "" + NameTabelle1 + ""."" + NameTabellenFeld1 +
""||"" + Feldtrenner + ""||"" + NameTabelle2 + ""."" + NameTabellenFeld2 + ""
FROM "" + NameTabelle1 + "", "" + NameTabelle2 + "" WHERE ""ID""=' + inID + ' AND
"" + NameTabelle1 + ""."" + NameTab12ID + ""='"" + NameTabelle2 + "".""ID""
ELSE
    stSql = "SELECT "" + NameTabelle2 + ""."" + NameTabellenFeld2 +
""||"" + Feldtrenner + ""||"" + NameTabelle1 + ""."" + NameTabellenFeld1 + ""
FROM "" + NameTabelle1 + "", "" + NameTabelle2 + "" WHERE ""ID""=' + inID + '
AND "" + NameTabelle1 + ""."" + NameTab12ID + ""='"" + NameTabelle2 + "".""ID""
END IF

```

Durch die Schreibweise in Basic ist der SQL-Befehl hier schon recht unübersichtlich. Jedes Feld und jeder Tabellename muss ja 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. Der Einfachheit halber geht diese Prozedur davon aus, dass alle Primärschlüsselfelder den Namen `"ID"` tragen.

Noch unübersichtlicher wird der Code dadurch, dass neben den durch doppelte Anführungsstriche eingefassten Tabellenfeldern auch noch Variablen eingefügt werden. Diese sind ja erst einmal keine Texte. Sie werden nur an den vorhergehenden Text durch ein `+` angehängt. Aber auch diese Variablen müssen maskiert werden. Dadurch erscheinen um diese Variablen sogar 3 Anführungsstriche oben: `""+NameTabelle1+""."" +NameTabellenFeld1+""` ergibt letztlich in der uns bekannten Abfragesprache `"Tabelle1"."Feld1"`. Zum Glück wird hier bei der Erstellung der Makros über den eingefärbten Code sichtbar, falls eventuell einmal ein doppeltes Anführungszeichen vergessen wurde. Die Anführungszeichen verändern sofort ihre Farbgebung, wenn sie nicht durch das Makro als Begrenzung von Zeichenketten erkannt werden.

```
ELSE
```

... und wenn eine zweite Tabelle nicht existiert, wird der folgende SQL-Code erstellt:

```

IF Position = 2 THEN
    stSql = "SELECT "" + NameTabellenFeld1 + ""||"" + Feldtrenner +
""||"" + NameTabellenFeld2 + "" FROM "" + NameTabelle1 + "" WHERE ""ID""=' +
inID + '"
ELSE
    stSql = "SELECT "" + NameTabellenFeld2 + ""||"" + Feldtrenner +
""||"" + NameTabellenFeld1 + "" FROM "" + NameTabelle1 + "" WHERE ""ID""=' +
inID + '"
END IF
END IF

```

ELSE

Wenn ein zweites Tabellenfeld nicht existiert kann es auch nur eine Tabelle sein. Daraus ergibt sich die folgende Abfrage:

```
        stSql = "SELECT "" + NameTabellenFeld1 + "" FROM "" + NameTabelle1 + ""  
WHERE ""ID""="" + inID + ""  
END IF
```

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

```
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)**'.

```
IF NOT IsNull(oAbfrageergebnis) THEN  
    WHILE oAbfrageergebnis.next  
        stFeldWert = oAbfrageergebnis.getString(1)  
    WEND
```

Das Kombinationsfeld muss jetzt auf den aus der Abfrage sich ergebenden Textwert eingestellt werden. Hierzu ist ein Zugriff auf den Controller notwendig.

```
oDocCr1 = ThisComponent.getCurrentController()  
oCtlView = oDocCr1.GetControl(oFeldList)  
oCtlView.setText(stFeldWert)  
END IF  
END IF
```

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.

```
IF IsEmpty(oFeld.getCurrentValue()) THEN  
    oDocCr1 = ThisComponent.getCurrentController()  
    oCtlView = oDocCr1.GetControl(oFeldList)  
    oCtlView.setText("")  
END IF  
END SUB
```

Diese Prozedur erledigt jetzt also den Kontakt von dem in einem versteckten Feld 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.

Übertragen eines Fremdschlüsselwertes vom Kombinationsfeld zum numerischen Feld

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

```
SUB TextAuswahlWertSpeichern(NameFormular AS STRING, NameUnterformular AS STRING,  
NameSubUnterformular AS STRING, NameFeld AS STRING, NameIDFeld AS STRING,  
NameTabellenFeld1 AS STRING, NameTabellenFeld2 AS STRING, Feldtrenner AS STRING,  
NameTabelle1 AS STRING, OPTIONAL NameTabelle2 AS STRING, OPTIONAL NameTab12ID AS  
STRING, OPTIONAL Position AS INTEGER )
```

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

Der Start dieser Funktion deckt sich vom Prinzip her mit dem der vorher geschilderten Prozedur. Auch hier gibt es Variablen, die optional sein sollen.

```

IF isMissing(NameTabelle2) THEN NameTabelle2 = ""
IF isMissing(NameTab12ID) THEN NameTab12ID = ""
IF isMissing(Position) THEN Position = 2

```

Danach werden die weiteren Variablen deklariert, die vorher noch nicht außerhalb einer Prozedur oder Funktion deklariert wurden. Dann wird das Formular angesteuert und die entsprechenden Felder mit Variablen belegt. Das Feld 'oFeld' enthält den Fremdschlüssel, das Feld 'oFeld-List' zeigt den Text dazu an.

```

DIM oForm AS OBJECT
DIM oSubForm AS OBJECT
DIM oSubSubForm AS OBJECT
DIM oFeld AS OBJECT
DIM oFeldList AS OBJECT
DIM stInhalt AS STRING
DIM stInhaltFeld2 AS STRING
DIM a_stTeile() AS STRING
DIM stmsgbox1 AS STRING
DIM stmsgbox2 AS STRING
DIM inID1 AS INTEGER
DIM inID2 AS INTEGER
DIM LaengeFeld1 AS INTEGER
DIM LaengeFeld2 AS INTEGER

```

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

```

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

```

Das Formular wird angesteuert, und das Kombinationsfeld ausgelesen.

```

oDoc = thisComponent
oDrawpage = oDoc.Drawpage
oForm = oDrawpage.Forms.getByName(NameFormular)
IF NameUnterformular <> "" THEN
  oSubForm = oForm.getByName(NameUnterformular)
  IF NameSubUnterformular <> "" THEN
    oSubSubForm = oSubForm.getByName(NameSubUnterformular)
    oFeld = oSubSubForm.getByName(NameIDFeld)
    oFeldList = oSubSubForm.getByName(NameFeld)
  ELSE
    oFeld = oSubForm.getByName(NameIDFeld)
    oFeldList = oSubForm.getByName(NameFeld)
  END IF
ELSE
  oFeld = oForm.getByName(NameIDFeld)
  oFeldList = oForm.getByName(NameFeld)
END IF
stInhalt = oFeldList.getCurrentValue()

```

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

```

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

```

Wenn ein zweites Tabellenfeld existiert muss der Inhalt des Kombinationsfeldes gesplittet werden. Um zu wissen, an welcher Stelle die Aufteilung vorgenommen werden soll, ist der Feldtrenner von Bedeutung, der der Funktion als Variable mitgegeben wird.

```
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.

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

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

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

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

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

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

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

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

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

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

Liegt kein zu langer Text vor, so kann die Funktion weiter durchlaufen. Ansonsten endet sie hier. Jetzt werden die Inhaltseingaben so maskiert, dass eventuell vorhandene Hochkommata keine Fehlermeldung erzeugen.

```

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

```

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

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

```

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

```

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

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

```

stSql = "SELECT ""ID"" FROM "" + NameTabelle2 + "" WHERE "" +
NameTabellenFeld2 + ""="" + stInhaltFeld2 + ""
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
IF NOT IsNull(oAbfrageergebnis) THEN
    WHILE oAbfrageergebnis.next
        inID2 = oAbfrageergebnis.getInt(1)
    WEND
END IF
IF inID2 = -1 THEN
    stSql = "INSERT INTO "" + NameTabelle2 + "" ("" +
NameTabellenFeld2 + """) VALUES (' + stInhaltFeld2 + ''') "
oSQL_Anweisung.executeUpdate(stSql)
stSql = "CALL IDENTITY()"

```

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

```

oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
IF NOT IsNull(oAbfrageergebnis) THEN
    WHILE oAbfrageergebnis.next
        inID2 = oAbfrageergebnis.getInt(1)
    WEND
END IF
END IF

```

Der Primärschlüssel aus dem zweiten Wert wird in der Variablen 'inID2' zwischengespeichert. Jetzt wird überprüft, ob eventuell dieser Schlüsselwert bereits in der Tabelle 1 zusammen mit dem

Eintrag aus dem ersten Feld vorhanden ist. Ist diese Kombination nicht vorhanden, so wird sie neu eingefügt.

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

```
stSql = "SELECT ""ID"" FROM "" + NameTabelle1 + "" WHERE "" +
NameTab12ID + ""="" + inID2 + "" AND "" + NameTabellenFeld1 + "" = '' + stInhalt +
""

oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
IF NOT ISNULL(oAbfrageergebnis) THEN
  WHILE oAbfrageergebnis.next
    inID1 = oAbfrageergebnis.getInt(1)
  WEND
END IF
```

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

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

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

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

```
stSql = "CALL IDENTITY()"
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
IF NOT IsNull(oAbfrageergebnis) THEN
  WHILE oAbfrageergebnis.next
    inID1 = oAbfrageergebnis.getInt(1)
  WEND
END IF
END IF
```

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

```
IF NameTabellenFeld2 <> "" AND NameTabelle2 = "" THEN
  stSql = "SELECT ""ID"" FROM "" + NameTabelle1 + "" WHERE "" +
NameTabellenFeld1 + ""="" + stInhalt + "" AND "" + NameTabellenFeld2 + ""="" +
stInhaltFeld2 + ""
  oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
IF NOT IsNull(oAbfrageergebnis) THEN
  WHILE oAbfrageergebnis.next
    inID1 = oAbfrageergebnis.getInt(1)
  WEND
END IF
IF inID1 = -1 THEN
```

... und eine zweite Tabelle nicht existiert:

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

Anschließend wird das Primärschlüsselfeld wieder ausgelesen.

```

        stSql = "CALL IDENTITY()"
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        IF NOT IsNull(oAbfrageergebnis) THEN
            WHILE oAbfrageergebnis.next
                inID1 = oAbfrageergebnis.getInt(1)
            WEND
        END IF
    END IF
END IF
IF NameTabellenFeld2 = "" THEN

```

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

```

        stSql = "SELECT ""ID"" FROM "" + NameTabelle1 + "" WHERE "" +
NameTabellenFeld1 + ""="" + stInhalt + ""
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        IF NOT IsNull(oAbfrageergebnis) THEN
            WHILE oAbfrageergebnis.next
                inID1 = oAbfrageergebnis.getInt(1)
            WEND
        END IF
        IF inID1 = -1 THEN

```

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

```

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

```

... und die entsprechende ID direkt wieder ausgelesen.

```

        stSql = "CALL IDENTITY()"
        oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
        IF NOT ISNULL(oAbfrageergebnis) THEN
            WHILE oAbfrageergebnis.next
                inID1 = oAbfrageergebnis.getInt(1)
            WEND
        END IF
    END IF
END IF

```

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

Anschließend wird der aus all diesen Schleifen ermittelte Primärschlüsselwert in das unsichtbare Feld der Haupttabelle und die darunterliegende Datenbank übertragen. Mit '**BoundField**' wird das mit dem Formularfeld verbundene Tabellenfeld erreicht. Mit '**updateInt**' wird eine Integer-Zahl (siehe Zahlendefinition) diesem Feld zugewiesen.

```

        oFeld.BoundField.updateInt(inID)
    END IF
ELSE

```

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

```

        oFeld.BoundField.updateNull()
    END IF
END SUB

```

Kontrollfunktion für die Zeichenlänge der Kombinationsfelder

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


```

FUNCTION Spaltengroesse(Tabellenname AS STRING, Feldname AS STRING) AS INTEGER
    oDatenquelle = ThisComponent.Parent.CurrentController
    If NOT (oDatenquelle.isConnected()) Then
        oDatenquelle.connect()
    End If
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "SELECT ""COLUMN_SIZE"" FROM ""INFORMATION_SCHEMA"". ""SYSTEM_COLUMNS""
    WHERE ""TABLE_NAME"" = ' " + Tabellenname + "' AND ""COLUMN_NAME"" = ' " + Feldname +
    ""'

    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)
    IF NOT IsNull(oAbfrageergebnis) THEN
        WHILE oAbfrageergebnis.next
            i = oAbfrageergebnis.getInt(1)
        WEND
    END IF
    Spaltengroesse = i
END FUNCTION

```

Aufruf der Prozedur zum Anzeigen des Textes

Die Prozedur zum Einstellen des Kombinationsfeldes wird bei jedem Datensatzwechsel aufgerufen. Das folgende Beispiel zeigt dies für die beigeigte Datenbank.

```

SUB Formular_Leser_Aufnahme_Start
    REM TextAnzeigen(NameFormular AS STRING, NameSubFormular AS STRING,
    NameSubSubFormular AS STRING, NameFeld AS STRING, NameIDFeld AS STRING,
    NameTabellenFeld1(aus Tabelle 1) AS STRING, NameTabellenFeld2(aus Tabelle 1 oder aus
    Tabelle 2) AS STRING, Feldtrenner AS STRING, NameTabelle1 AS STRING, OPTIONAL
    NameTabelle2 AS STRING, OPTIONAL NameIDAusTabelle2InTabelle1 AS STRING, OPTIONAL
    Position des Feldes aus Tabelle2 in der Combobox AS INTEGER [1 oder 2] )
    TextAnzeigen ("Filter", "Formular", "Adresse", "comStr", "numStrID", "Strasse",
    "", "", "Strasse")
    TextAnzeigen ("Filter", "Formular", "Adresse", "comPlzOrt", "numPlzOrtID",
    "Postleitzahl", "Ort", " ", "Postleitzahl", "Ort", "Ort_ID", 2)
END SUB

```

Der Inhalt dieser als Kommentar beigefügten Zeilen sagt eigentlich schon, was einzutragen ist. Die Prozedur wird mit dieser Reihe von Parametern versorgt. Hat ein Parameter keinen Inhalt, so werden stattdessen einfach zwei doppelte Anführungsstriche "" weitergegeben. Die letzten drei Parameter dürfen ausgelassen werden, da sie in der Prozedur gegebenenfalls mit Standardwerten belegt werden.

Es wird also die Prozedur '**TextAnzeigen**' aufgerufen. Das Formular, in dem die Felder liegen, ist das Formular **Filter** → **Formular** → **Adresse**. Es handelt sich also um das Unterformular eines Unterformulars.

Das erste Kombinationsfeld, in dem die Straße eingegeben wird, heißt '**comStr**', das versteckte Fremdschlüsselfeld für die dem Formular zugrundeliegende Tabelle heißt '**numStrID**'. In dem ersten Kombinationsfeld wird das Feld '**Strasse**' angezeigt. Die Tabelle, in der die Einträge des Kombinationsfeld stehen sollen, hat die Bezeichnung '**Strasse**'.

Im zweiten Kombinationsfeld wird die Postleitzahl und der Ort eingegeben. Es hat die Bezeichnung '**comPlzOrt**'. Das versteckte Fremdschlüsselfeld hat die Bezeichnung '**numPlzOrtID**'. In dem zweiten Kombinationsfeld werden die Felder '**Postleitzahl**' und '**Ort**', getrennt durch eine Leertaste (" "), wiedergegeben. Die erste Tabelle hat den Namen '**Postleitzahl**', die zweite Tabelle den Namen '**Ort**'. In der ersten Tabelle lautet der Fremdschlüssel der zweiten Tabelle '**Ort_ID**'. Das Feld der zweiten Tabelle wird als zweites in dem Kombinationsfeld angezeigt, also Position 2.

Aufruf der Prozedur zur Textspeicherung

Beim Speichern wird die Prozedur TextAuswahlWertSpeichern aufgerufen.

SUB Formular_Leser_Ausleihe_Speichern

```
REM TextAuswahlWertSpeichern(NameFormular AS STRING, NameSubFormular AS STRING,
NameSubSubFormular AS STRING, NameFeld AS STRING, NameIDFeld AS STRING, NameTabellen-
Feld1(aus Tabelle 1) AS STRING, NameTabellenFeld2(aus Tabelle 1 oder aus Tabelle 2)
AS STRING, Feldtrenner AS STRING, NameTabelle1 AS STRING, OPTIONAL NameTabelle2 AS
STRING, OPTIONAL NameIDAusTabelle2InTabelle1 AS STRING, OPTIONAL Position des Feldes
aus Tabelle2 in der Combobox AS INTEGER [1 oder 2] )
```

Der Kommentar ist der gleiche wie bei der vorherigen Prozedur. Entsprechend stimmen auch die übergebenen Variablen überein.

```
TextAuswahlWertSpeichern ("Filter", "Formular", "Adresse", "comStr", "numStrID",
"Strasse", "", "", "Strasse", "", "")
TextAuswahlWertSpeichern ("Filter", "Formular", "Adresse", "comPlzOrt",
"numPlzOrtID", "Postleitzahl", "Ort", " ", "Postleitzahl", "Ort", "Ort_ID", 2)
END SUB
```

Damit der Wert auch sicher abgespeichert wird, muss dem Formular eine Änderung mitgeteilt werden. Schließlich erfolgt die Änderung für den Benutzer nur in dem Kombinationsfeld, während das Formular keine weitere Verbindung zum Kombinationsfeld hat. Mit der Datenbank ist vielmehr das numerische Feld für den Fremdschlüssel verbunden. So wird einfach dem Fremdschlüsselfeld der Wert -1 zugewiesen, den ja ein Autowertfeld nicht einnehmen kann. Damit ist dann auf jeden Fall eine Änderung des Feldinhaltes verbunden. Dieser Feldinhalt wird anschließend durch die Abarbeitung des Listenfeldinhaltes wieder überschrieben.

SUB Datensatzaktion_erzeugen(oEvent AS OBJECT)

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

```
DIM oForm AS OBJECT
DIM oSubForm AS OBJECT
DIM oSubSubForm AS OBJECT
DIM oFeld AS OBJECT
DIM stTag AS String
stTag = oEvent.Source.Model.Tag
aForms() = Split(stTag, ",")
```

Ein Array wird gegründet, der Feldname steht zuerst, die Formularnamen vom Hauptformular zum ersten und zweiten Unterformular danach.

```
oDoc = thisComponent
oDrawpage = oDoc.Drawpage
oForm = oDrawpage.Forms.getByName(aForms(1))
IF UBound(aForms()) > 1 THEN
    oForm = oForm.getByName(aForms(2))
    IF UBound(aForms()) > 2 THEN
        oForm = oForm.getByName(aForms(3))
    END IF
END IF
oFeld = oForm.getByName(aForms(0))
oFeld.BoundField.updateInt(-1)
END SUB
```

Navigation von einem Formular zum anderen

Ein Formular soll über ein entsprechendes Ereignis geöffnet werden.

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

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

Das Array wird gegründet und mit den Formularnamen gefüllt, in diesem Fall zuerst in dem zu öffnenden Formular und als zweites dem aktuellen Formular, dass nach dem Öffnen des anderen geschlossen werden soll.

```

ThisDatabaseDocument.FormDocuments.getByname( Trim(aForm(0)) ).open
ThisDatabaseDocument.FormDocuments.getByname( Trim(aForm(1)) ).close
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:

```

SUB Hauptformular_oeffnen
ThisDatabaseDocument.FormDocuments.getByname( "Hauptformular" ).open
END SUB

```

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

```

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

```

Formulare dokumente, die in einem Verzeichnis liegen, werden in den Zusatzinformationen als Verzeichnis/Formular angegeben. Dies muss umgewandelt werden zu `...getbyname("Verzeichnis").getbyname("Formular")`.

Datenbankaufgaben mit Makros erweitert

Verbindung mit Datenbanken erzeugen

```

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

```

Hier wäre es möglich, fest einen Benutzernamen und ein Passwort einzugeben, wenn eine Passworteingabe erforderlich wäre. In den Klammern steht dann ("Benutzername","Passwort"). Statt einen Benutzernamen und ein Passwort in Reinschrift einzutragen, wird für diesen Fall der Dialog für den Passwortschutz aufgerufen:

```

ELSE

```

```

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

```

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

```

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

```

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

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

```

oDatenquelle = ThisComponent.Drawpage.Forms(0)
oVerbindung = oDatenquelle.activeConnection

```

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 der Beendigung der Base-Datei erfolgt. Schließlich wird in diesem Moment der Inhalt der Datenbank in die Datei zurückgeschrieben.

Außerdem gibt es die üblichen Verdächtigen für plötzlich nicht mehr zu öffnende Dateien wie Festplattenfehler usw. Da kann es dann nicht schaden, eine Sicherheitskopie möglichst mit dem aktuellsten Datenstand parat zu haben. Der Datenbestand ändert sich allerdings nicht, während die *.odb-Datei geöffnet ist. Deshalb kann die Sicherung direkt mit dem Öffnen der Datei verbunden werden. Es werden einfach Kopien der Datei in das unter Extras → Optionen → LibreOffice → Pfade angegebene Backup-Verzeichnis erstellt. Dabei beginnt das Makro nach 5 Kopien damit, die jeweils älteste Variante zu überschreiben.

```

SUB Datenbankbackup
DIM oPath AS OBJECT
DIM oDoc AS OBJECT
DIM sTitel AS STRING
DIM sUrl_Ziel AS STRING
DIM sUrl_Start AS STRING
DIM i AS INTEGER
DIM k AS INTEGER
oDoc = ThisComponent
sTitel = oDoc.Title
sUrl_Start = oDoc.URL
oPath = createUnoService("com.sun.star.util.PathSettings")
FOR i = 1 TO 6
IF NOT FileExists(oPath.Backup & "/" & i & "_" & sTitel) THEN
IF i > 5 THEN
FOR k = 4 TO 1 STEP -1
IF FileDateTime(oPath.Backup & "/" & k & "_" & sTitel) <=
FileDateTime(oPath.Backup & "/" & k+1 & "_" & sTitel) THEN
IF k = 1 THEN
i = k
EXIT FOR
END IF
ELSE
i = k + 1
EXIT FOR
END IF

```

```

        NEXT
    END IF
    EXIT FOR
END IF
NEXT
sUrl_Ziel = oPath.Backup & "/" & i & "_" & sTitel
FileCopy(sUrl_Start,sUrl_Ziel)
END SUB

```

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

```

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

```

Datenbanken komprimieren

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

Wird die Komprimierung durchgeführt, so kann anschließend nicht mehr auf die Tabellen zugegriffen werden. Die Datei muss neu geöffnet werden. Deshalb schließt das Makro auch das Formular, aus dem heraus es aufgerufen wird. Leider lässt sich das Dokument selbst nicht schließen, ohne dass beim Neuaufruf die Wiederherstellung anspringt. Deshalb ist diese Funktion auskommentiert.

```

SUB Datenbank_komprimieren
    DIM stMessage AS STRING
    oDatenquelle = ThisComponent.Parent.CurrentController ' Zugriffsmöglichkeit aus dem
Formular heraus
    IF NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "SHUTDOWN COMPACT" ' Die Datenbank wird komprimiert und geschlossen
    oSQL_Anweisung.executeQuery(stSql)
    stMessage = "Die Datenbank wurde komprimiert." + CHR(13) + "Das Formular wird
jetzt geschlossen."
    stMessage = stMessage + CHR(13) + "Anschließend sollte die Datenbankdatei
geschlossen werden."
    stMessage = stMessage + CHR(13) + "Auf die Datenbank kann erst nach dem erneuten
Öffnen der Datenbankdatei zugegriffen werden."
    msgbox stMessage
    ThisDatabaseDocument.FormDocuments.getByName( "Wartung" ).close
    REM Das Schließen der Datenbankdatei führt beim Neustart zu einem
Wiederherstellungsablauf.
    ' ThisDatabaseDocument.close(True)
END SUB

```

Tabellenindex heruntersetzen bei Autowert-Feldern

Werden viele Daten aus Tabellen gelöscht, so stören sich Nutzer häufig daran, dass die automatisch erstellten Primärschlüssel einfach weiter hochgezählt werden, statt direkt an den bisher höchsten Schlüsselwert anzuschließen. Die folgende Prozedur liest für eine Tabelle den bisherigen

Höchstwert des Feldes "ID" aus und stellt den nächsten Schlüsselstartwert um 1 höher als das Maximum ein.

Heißt das Primärschlüsselfeld nicht "ID", so müsste das Makro entsprechend angepasst werden.

```
SUB Tabellenindex_runter(stTabelle AS STRING)
    REM Mit dieser Prozedur wird das automatisch hochgeschriebene Primärschlüsselfeld
    mit der vorgegebenen Bezeichnung "ID" auf den niedrigst möglichen Wert eingestellt.
    DIM inAnzahl AS INTEGER
    DIM inSequence_Value AS INTEGER
    oDatenquelle = ThisComponent.Parent.CurrentController ' Zugriffsmöglichkeit aus dem
    Formular heraus
    IF NOT (oDatenquelle.isConnected()) THEN
        oDatenquelle.connect()
    END IF
    oVerbindung = oDatenquelle.ActiveConnection()
    oSQL_Anweisung = oVerbindung.createStatement()
    stSql = "SELECT MAX("ID") FROM ""+stTabelle+"" ' Der höchste in "ID"
    eingetragene Wert wird ermittelt
    oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql) ' Abfrage starten und den
    Rückgabewert in einer Variablen oAbfrageergebnis speichern
    IF NOT ISNULL(oAbfrageergebnis) THEN
        WHILE oAbfrageergebnis.next
            inAnzahl = oAbfrageergebnis.getInt(1) ' Erstes Datenfeld wird ausgelesen
        WEND ' nächster Datensatz, in diesem Fall nicht mehr erforderlich, da nur ein
        Datensatz existiert
        IF inAnzahl = "" THEN ' Falls der höchste Wert gar kein Wert ist, also die
        Tabelle leer ist wird der höchste Wert als -1 angenommen
            inAnzahl = -1
        END IF
        inSequence_Value = inAnzahl+1 ' Der höchste Wert wird um 1 erhöht
        REM Ein neuer Befehl an die Datenbank wird vorbereitet. Die ID wird als neu
        startend ab inAnzahl+1 deklariert.
        REM Diese Anweisung hat keinen Rückgabewert, da ja kein Datensatz ausgelesen
        werden muss
        oSQL_Anweisung1 = oVerbindung.createStatement()
        oSQL_Anweisung1.executeQuery("ALTER TABLE "" + stTabelle + "" ALTER COLUMN
        ""ID"" RESTART WITH " + inSequence_Value + "")
    END IF
END SUB
```

Serienbriefdruck aus Base heraus

Manchmal reicht einfach ein Bericht nicht aus, um sauber Briefe an die Adressaten zu erstellen. Schon die Textfelder in einem Bericht sind hier in der Nutzung doch sehr eingeschränkt. Hierzu wird dann ein Serienbrief im Writer erstellt. Es muss aber nicht sein, dass erst der Writer geöffnet wird, dort dann über den Serienbriefdruck alle Auswahlmöglichkeiten und Eingaben gemacht werden und schließlich der Druck erfolgt. Dies alles geht auch per Makro direkt aus Base heraus.

```
SUB Serienbriefdruck
    DIM oMailMerge AS OBJECT
    DIM aProps()
    oMailMerge = createunoservice("com.sun.star.text.MailMerge")
```

Als Name der Datenquelle wird der Name angegeben, unter dem die Datenbank in LO angemeldet ist. Dieser Name muss nicht identisch mit dem Dateinamen sein. Der Anmeldenamen in diesem Beispiel lautet "Adressen"

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

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

```
oMailMerge.DocumentURL = ConvertToUrl("home/user/Dokumente/Serienbrief.odt")
```

Der Typ des Kommandos wird festgelegt. '0' steht für eine Tabelle, '1' für eine Abfrage und '2' für ein direktes SQL-Kommando.

```
oMailMerge.CommandType = 1
```

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

```
oMailMerge.Command = "Serienbriefabfrage"
```

Über den Filter wird festgelegt, für welche Datensätze aus der Serienbriefabfrage ein Druck erfolgen soll. Dieser Filter könnte z.B. über ein Formularfeld aus Base heraus an das Makro weiter gegeben werden. Mit dem Primärschlüssel eines Datensatzes könnte so der Ausdruck eines einzelnen Dokumentes erfolgen.

In diesem Beispiel wird aus der "Serienbriefabfrage" das Feld "Geschlecht" aufgesucht und dort nach Datensätzen gesucht, die in diesem Feld mit einem 'm' versehen sind.

```
oMailMerge.Filter = """Geschlecht""='m'""
```

Es gibt die Ausgabetypen Drucker (1), Datei (2) und Mail (3). Hier wurde zu Testzwecken die Ausgabe in eine Datei gewählt. Diese Datei wird in dem angegebenen Pfad abgespeichert. Für jeden Serienbriefdatensatz wird ein Druck erzeugt. Damit dieser Druck unterscheidbar ist, wird das Feld Nachname in den Dateinamen aufgenommen.

```
oMailMerge.OutputType = 2
oMailMerge.OutputUrl = ConvertToUrl("home/user/Dokumente")
oMailMerge.FileNameFromColumn = True
oMailMerge.FileNamePrefix = "Nachname"
oMailMerge.execute(aProps())
```

END SUB

Wird allein der Filter über ein Formular bestückt, so kann auf diese Art und Weise also ohne die Öffnung des Writer-Dokumentes ein Serienbriefdruck erfolgen.

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** «Mailstart_Dateiaufruf.odt»

```
SUB Link_oeffnen
DIM oDoc AS OBJECT
DIM oDrawpage AS OBJECT
DIM oForm AS OBJECT
DIM oFeld AS OBJECT
DIM oShell AS OBJECT
DIM stFeld AS STRING
oDoc = thisComponent
oDrawpage = oDoc.Drawpage
oForm = oDrawpage.Forms.getByName("Formular")
oFeld = oForm.getByName("Adresse")
```

Aus dem benannten Feld wird der Inhalt ausgelesen. Dies kann eine Webadresse, beginnend mit 'http://', eine Mailadresse mit einem '@' oder ein einfaches Dokument sein, das durch eine entsprechende Pfadangabe aufgesucht werden soll (z.B. externe Bilder, *.pdf-Dateien, Tondokumente ...).

```
stFeld = oFeld.Text
IF stFeld = "" THEN
    EXIT SUB
END IF
```

Ist das Feld leer, so soll das Makro sofort enden. Bei der Eingabe passiert es ja sehr oft, dass Felder mit der Maus aufgesucht werden. Ein Mausklick in das Feld, um dort zu ersten Mal schreiben zu können, soll aber noch nicht das Makro ausführen.

Jetzt wird gesucht, ob in dem Feld ein '@' enthalten ist. Dies deutet auf eine E-Mail-Adresse hin. Es soll also das Mailprogramm gestartet werden und eine Mail an diese Mailadresse gesandt werden.

```
IF InStr(stFeld,"@") THEN
    stFeld = "mailto:"+stFeld
```

Ist kein '@' vorhanden, so wird der Begriff so konvertiert, dass die Datei im Dateisystem gefunden werden kann. Steht ein 'http://' am Anfang, so wird bei dieser Funktion nicht im Dateisystem sondern über den Webbrowser direkt im Internet nachgesehen. Ansonsten beginnt der erstellte Pfad anschließend mit dem Begriff 'file:///'

```
ELSE
    stFeld = convertToUrl(stFeld)
END IF
```

Jetzt wird das Programm aufgesucht, das in dem eigenen Betriebssystem mit der entsprechenden Dateiendung verbunden ist. Bei dem Stichwort 'mailto:' ist dies das Mailprogramm, bei 'http://' der Webbrowser und bei allen anderen ist die Entscheidung des Systems mit den Endungen der Datei verbunden.

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

Aufruf eines Mailprogramms mit Inhaltvorgaben

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.odt»

Der Mailaufruf erfolgt mit 'mailto:Empfänger?subject= &body= &cc= &bcc= '. Die letzten beiden Eingaben sind im Formular allerdings nicht aufgeführt. Anhänge kommen in der Definition von 'mailto' nicht vor. Manchmal funktioniert allerdings 'attachment='.

```
SUB Mail_Aufruf
    DIM oDoc AS OBJECT
    DIM oDrawpage AS OBJECT
    DIM oForm AS OBJECT
    DIM oFeld1 AS OBJECT
    DIM oFeld2 AS OBJECT
    DIM oFeld3 AS OBJECT
    DIM oFeld4 AS OBJECT
    DIM oShell AS OBJECT
    DIM stFeld1 AS STRING
    DIM stFeld2 AS STRING
    DIM stFeld3 AS STRING
    DIM stFeld4 AS STRING
    oDoc = thisComponent
    oDrawpage = oDoc.Drawpage
    oForm = oDrawpage.Forms.getByName("Formular")
    oFeld1 = oForm.getByName("E-Mail-Adresse")
    oFeld2 = oForm.getByName("E-Mail-Betreff")
    oFeld3 = oForm.getByName("E-Mail-Inhalt")
    stFeld1 = oFeld1.Text
    IF stFeld1 = "" THEN
        msgbox "Keine Mailadresse vorhanden." & CHR(13) & "Das Mailprogramm wird nicht
aufgerufen" , 48, "Mail senden"
    END IF
END SUB
```


Die Konvertierung zu URL ist notwendig, damit Sonderzeichen und Zeilenumbrüche den Aufruf nicht stören. Dabei wird allerdings auch der Begriff 'file:/// ' vorangestellt. Diese 8 Zeichen zu Beginn werden nicht mit übernommen.

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

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

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

Mauszeiger beim Überfahren eines Links ändern

Im Internet üblich, bei Base nachgebaut: Der Mauszeiger fährt über ein Textfeld und verändert seine Form zu einer zeigenden Hand. Der enthaltene Text kann jetzt noch in den Eigenschaften des Feldes zu der Farbe Blau und unterstrichen geändert werden – schon ist der Eindruck eines Links aus dem Internet perfekt. Jeder Nutzer erwartet nach einem Klick, dass sich ein externes Programm öffnet.

Siehe auch zu diesem Abschnitt die **Beispieldatenbank** «Mailstart_Dateiaufruf.odt»

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

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

Formulare ohne Symbolleisten präsentieren

Neunutzer von Base sind häufig irritiert, dass z.B. eine Menüleiste existiert, diese aber im Formular so gar nicht verfügbar ist. Diese Menüleisten können auf verschiedene Arten ausgeblendet werden. Am erfolgreichsten unter allen LO-Versionen sind die beiden im Folgenden vorgestellten Vorgehensweisen.

Formulare ohne Symbolleisten in einem Fenster

Ein Fenster lässt sich in der Größe variieren. Über den entsprechenden Button lässt es sich auch schließen. Diese Aufgaben übernimmt der Window-Manager des jeweiligen Betriebssystems. Lage und Größe des Fensters auf dem Bildschirm kann beim Start über ein Makro mitgegeben werden.

```
SUB Symbolleisten_Ausblenden
DIM oFrame AS OBJECT
DIM oWin AS OBJECT
DIM oLayoutMng AS OBJECT
DIM aElemente()
oFrame = StarDesktop.getCurrentFrame()
```

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

```
oFrame.setTitle "Mein Formular"
oWin = oFrame.getContainerWindow()
```

Das Fenster wird auf die maximale Größe eingestellt. Dies entspricht nicht dem Vollbildmodus, da z.B. eine Kontrollleiste noch sichtbar ist und das Fenster eine Titelleiste hat, über die die Größe des Fensters geändert und das Fenster geschlossen werden kann.

```
oWin.IsMaximized = true
```

Es besteht auch die Möglichkeit, das Fenster in einer ganz bestimmten Größe und mit einer festen Position darzustellen. Dies würde mit `'oWin.setSize(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 auf mit den folgenden Werten über eine Summierung berechnet: $x=1, y=2, \text{Breite}=4, \text{Höhe}=8$. Da x, y, Breite und Höhe angegeben sind, hat das **'Flag'** die Größe $1 + 2 + 4 + 8 = 15$.

```
oLayoutMng = oFrame.LayoutManager
aElemente = oLayoutMng.getElements()
FOR i = LBound(aElemente) TO UBound(aElemente)
    IF aElemente(i).ResourceURL = "private:resource/toolbar/formsnavigationbar"
THEN
```

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.

Hier wäre es auch möglich, sie als unsichtbar zu deklarieren, Nur erscheint dann, je nach Office-Version, z.B. die Menüleiste nach einem Klick in ein Formularelement wieder und lässt sich nicht erneut als unsichtbar deklarieren.

```
ELSE
    msgbox aElemente(i).ResourceURL
    oLayoutMng.hideElement(aElemente(i).ResourceURL)
END IF
NEXT
END SUB
```

Werden die Symbolleisten nicht wieder direkt beim Beenden des Formulars eingeblendet, so bleiben sie weiterhin verborgen. Sie können natürlich über **Ansicht** → **Symbolleisten** wieder aufgerufen werden. Etwas irritierend ist es jedoch, wenn gerade die Standardleiste (**Ansicht** → **Symbolleisten** → **Standardleiste**) oder die Statusleiste (**Ansicht** → **Statusleiste**) fehlt.

Mit dieser Prozedur werden die Symbolleisten aus dem Versteck (`'hideElement'`) wieder hervorgeholt (`'showElement'`). Der Kommentar enthält die Leisten, die oben als sonst fehlende Leisten am ehesten auffallen.

```
SUB Symbolleisten_Einblenden
DIM oFrame AS OBJECT
DIM oLayoutMng AS OBJECT
DIM aElemente()
oFrame = StarDesktop.getCurrentFrame()
oLayoutMng = oFrame.LayoutManager
aElemente = oLayoutMng.getElements()
FOR i = LBound(aElemente) TO UBound(aElemente)
    oLayoutMng.showElement(aElemente(i).ResourceURL)
    ' eventuell fehlende wichtige Elemente:
    ' "private:resource/toolbar/standardbar"
    ' "private:resource/statusbar/statusbar"
NEXT
END SUB
```

Formulare im Vollbildmodus

Beim Vollbildmodus wird der gesamte Bildschirm vom Formular bedeckt. Hier steht keine Kontrollleiste o.ä. mehr zur Verfügung, die gegebenenfalls anzeigt, ob noch irgendwelche anderen Programme laufen.

```
FUNCTION Fullscreen(boSwitch AS BOOLEAN)
DIM oDispatcher AS OBJECT
DIM Props(0) AS NEW com.sun.star.beans.PropertyValue
oDispatcher = createUnoService("com.sun.star.frame.DispatchHelper")
Props(0).Name = "FullScreen"
Props(0).Value = boSwitch
```

```

oDispatcher.executeDispatch(ThisComponent.CurrentController.Frame,
".uno:FullScreen", "", 0, Props())
END FUNCTION

```

Diese Funktion wird durch die folgenden Prozeduren eingeschaltet. In den Prozeduren läuft gleichzeitig die vorhergehende Prozedur zum Ausblenden der Symbolleisten ab – sonst erscheint die Symbolleiste, mit der der Vollbildmodus wieder ausgeschaltet werden kann. Auch dies ist eine Symbolleiste, wenn auch nur mit einem Symbol.

```

SUB Vollbild_ein
Fullscreen(true)
Symbolleisten_Ausblenden
END SUB

```

Aus dem Vollbild-Modus geht es wieder heraus über die 'ESC'-Taste. Wenn stattdessen ein Button mit einem entsprechenden Befehl belegt werden soll, so reichen auch die folgenden Zeilen:

```

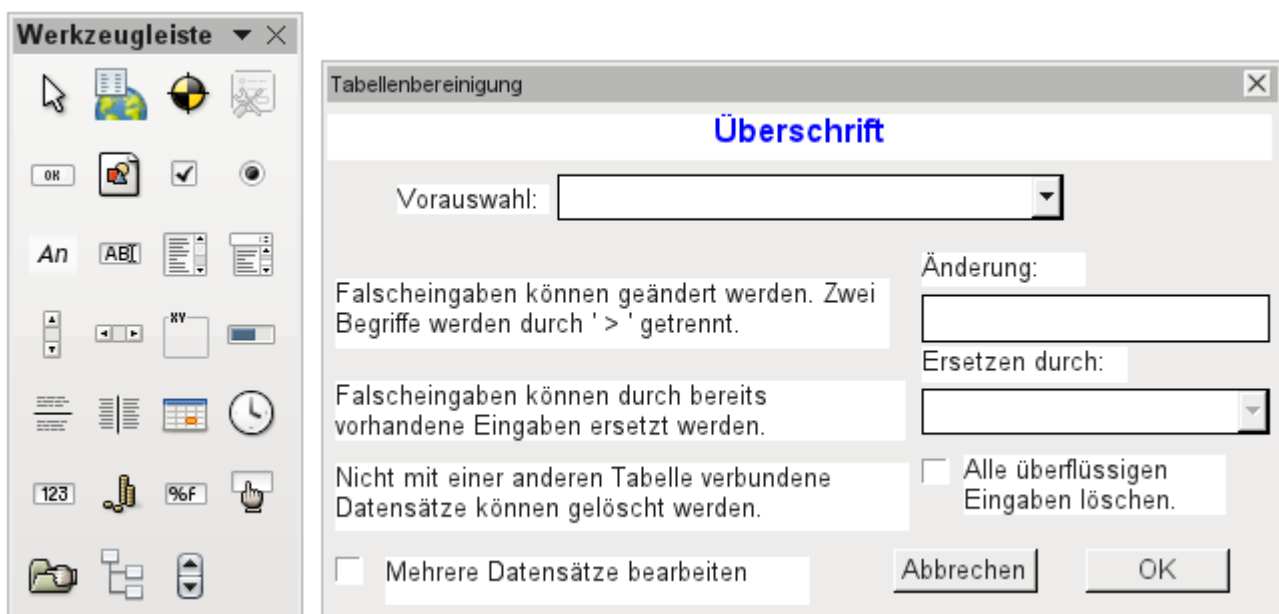
SUB Vollbild_aus
Fullscreen(false)
Symbolleisten_Einblenden
END SUB

```

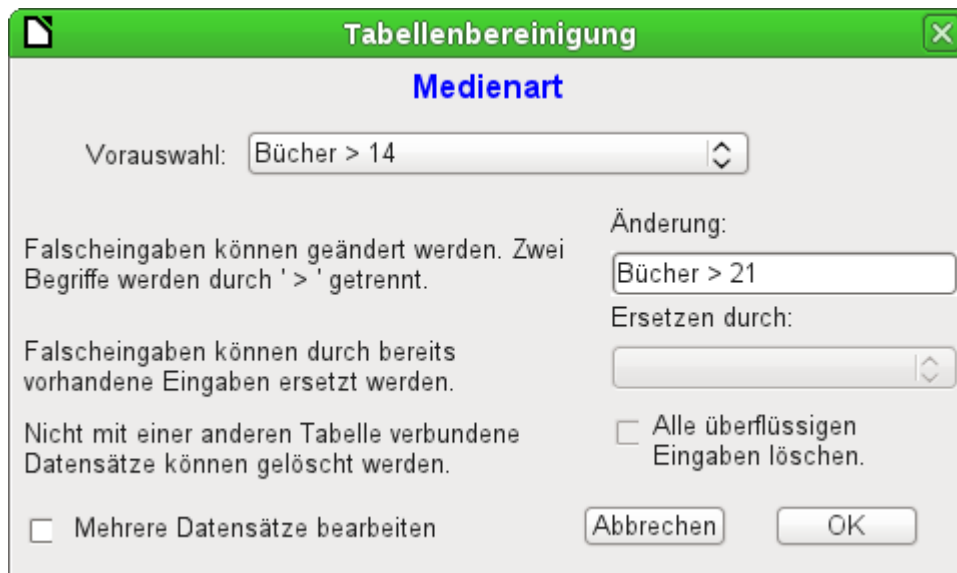
Dialoge

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.



Dialoge werden neben den Modulen für Makros abgespeichert. Ihre Erstellung erfolgt ähnlich der eines Formulars. Hier stehen auch weitgehend ähnliche Kontrollfelder zur Verfügung. Lediglich das Tabellenkontrollfeld aus dem Formular fehlt als besondere Eingabemöglichkeit.



Wird ein Dialog ausgeführt, so erscheinen die Kontrollfelder entsprechend der Einstellung der grafischen Benutzeroberfläche.

Der oben abgebildete Dialog der Beispieldatenbank soll dazu dienen, die Tabellen zu bearbeiten, die nicht direkt in einem der Formulare als Grundlage vorhanden sind. So ist z.B. die Medienart über ein Listenfeld zugänglich, in der Makro-Version bereits durch ein Kombinationsfeld. In der Makro-Version können die Inhalte der Felder zwar durch neue Inhalte ergänzt werden, eine Änderung alter Inhalte ist aber nicht möglich. In der Version ohne Makros erfolgt die Änderung über ein separates Tabellenkontrollfeld.

Während die Änderung noch ohne Makros recht einfach in den Griff zu bekommen ist, so ist es doch recht umständlich, die Medienart vieler Medien auf eine andere Medienart zu ändern. Angenommen, es gäbe die Medienarten 'Buch, gebunden', 'Buch, kartoniert', 'Taschenbuch' und 'Ringbuch'. Jetzt stellt sich nach längerem Betrieb der Datenbank heraus, dass muntere Zeitgenossen noch weitere ähnliche Medienarten für gedruckte Werke vorgesehen haben. Nur ist uns die Differenzierung viel zu weitgehend. Es soll also reduziert werden, am liebsten auf nur einen Begriff. Ohne Makro müssten jetzt die Datensätze in der Tabelle Medien (mit Hilfe von Filtern) aufgesucht werden und einzeln geändert werden. Mit Kenntnis von SQL geht dies über die SQL-Eingabe schon wesentlich besser. Mit einer Eingabe werden alle Datensätze der Tabelle Medien geändert. Mit einer zweiten SQL-Anweisung wird dann die jetzt überflüssige Medienart gelöscht, die keine Verbindung mehr zur Tabelle "Medien" hat. Genau dieses Verfahren wird mit diesem Dialog über «Ersetzen durch:» angewandt – nur dass eben die SQL-Anweisung erst über das Makro an die Tabelle "Medienart" angepasst wird, da das Makro auch andere Tabellen bearbeiten können soll.

Manchmal schleichen sich auch Eingaben in eine Tabelle ein, die im Nachhinein in den Formularen geändert wurden, also eigentlich gar nicht mehr benötigt werden. Da kann es nicht schaden, solche verwaisten Datensätze einfach zu löschen. Nur sind die über die grafische Oberfläche recht schwer ausfindig zu machen. Hier hilft wieder eine entsprechende SQL-Abfrage, die mit einer Löschanweisung gekoppelt ist. Diese Anweisung ist im Dialog je nach betroffener Tabelle unter «Alle überflüssigen Eingaben löschen» hinterlegt.

Sollen mit dem Dialog mehrere Änderungen durchgeführt werden, so ist dies über das Markierfeld «Mehrere Datensätze bearbeiten» anzugeben. Dann endet der Dialog nicht mit der Betätigung des Buttons «OK».

Der Makrocode für diesen Dialog ist aus der Beispieldatenbank ersichtlich. Im Folgenden werden nur Ausschnitte daraus erläutert.

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

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

0: Formular, 1: Unterformular, 2: UnterUnterformular, 3: Kombinationsfeld oder Tabellenkontrollfeld, 4: Fremdschlüsselfeld im Formular, bei Tabellenkontrollfeld leer, 5: Tabellename Nebentabelle, 6: Tabellenfeld1 Nebentabelle, 7: Tabellenfeld2 Nebentabelle, ggf. 8: Tabellename Nebentabelle für Tabellenfeld 2

Die Einträge in diesem Bereich werden zu Beginn des Makros als Kommentar aufgelistet. Die damit verbundenen Ziffern geben die Ziffern wieder, unter denen der jeweilige Eintrag aus dem Array ausgelesen wird. Das Makro kann Listenfelder verarbeiten, die zwei Einträge, getrennt durch «>», enthalten. Diese beiden Einträge können auch aus unterschiedlichen Tabellen stammen und über eine Abfrage zusammengeführt sein, wie z.B. bei der Tabelle "Postleitzahl", die für die Orte lediglich das Fremdschlüsselfeld "Ort_ID" enthält, zur Darstellung des Ortes also die Tabelle "Ort" benötigt.

```
DIM aFremdTabellen(0, 0 to 1)
DIM aFremdTabellen2(0, 0 to 1)
```

Unter den zu Beginn definierten Variablen fallen zwei Arrays auf. Während normale Arrays auch durch den Befehl '**Split()**' während der Laufzeit der Prozedur erstellt werden können, müssen zweidimensionale Arrays vorher definiert werden. Zweidimensionale Arrays werden z.B. benötigt, um aus einer Abfrage mehrere Datensätze zu speichern, bei denen die Abfrage selbst über mehr als ein Feld geht. Die beiden obigen Arrays müssen Abfragen auswerten, die sich jeweils auf zwei Tabellenfelder beziehen. Deshalb werden sie in der zweiten Dimension mit '0 to 1' auf zwei unterschiedliche Inhalte festgelegt.

```
stTag = oEvent.Source.Model.Tag
aTabelle() = Split(stTag, ", ")
FOR i = LBound(aTabelle()) TO UBound(aTabelle())
    aTabelle(i) = trim(aTabelle(i))
NEXT
```

Die mitgegebenen Variablen werden ausgelesen. Die Reihenfolge steht im obigen Kommentar. Es gibt maximal 9 Einträge, wobei geklärt werden muss, ob ein 8. Eintrag für das Tabellenfeld2 und ein 9. Eintrag für eine zweite Tabelle existieren.

Wenn Werte aus einer Tabelle entfernt werden, so muss zuerst einmal berücksichtigt werden, ob sie nicht noch als Fremdschlüssel in anderen Tabellen existieren. In einfachen Tabellenkonstruktionen gibt es von einer Tabelle aus lediglich eine Fremdschlüsselverbindung zu einer anderen Tabelle. In der vorliegenden Beispieldatenbank aber wird z.B. die Tabelle "Ort" genutzt, um die Erscheinungsorte der Medien und die Orte für die Adressen zu speichern. Es wird also zweimal der Primärschlüssel der Tabelle "Ort" in unterschiedlichen Tabellen eingetragen. Diese Tabellen und Fremdschlüsselbezeichnungen könnten natürlich auch über die «Zusatzinformationen» eingegeben werden. Schöner wäre es aber, wenn sie universell für alle Fälle ermittelt werden. Dies geschieht durch die folgende Abfrage.

```
stSql = "SELECT ""FKTABLE_NAME"", ""FKCOLUMN_NAME"" FROM
""INFORMATION_SCHEMA"". ""SYSTEM_CROSSREFERENCE"" WHERE ""PKTABLE_NAME"" = ' " +
aTabelle(5) + "'"
```

In der Datenbank sind im Bereich "INFORMATION_SCHEMA" alle Informationen zu den Tabellen der Datenbank abgespeichert, so auch die Informationen zu den Fremdschlüsseln. Die entsprechende Tabelle, die diese Informationen enthält, ist über "INFORMATION_SCHEMA"."SYSTEM_CROSSREFERENCE" erreichbar. Mit "PKTABLE_NAME" wird die Tabelle erreicht, die ihren Primärschlüssel ("Primary Key") in die Beziehung mit einbringt. Mit "FKTABLE_NAME" wird die Tabelle erreicht, die diesen Primärschlüssel als Fremdschlüssel ("Foreign Key") nutzt. Über "FKCOLUMN_NAME" wird schließlich die Bezeichnung des Fremdschlüsselfeldes ermittelt.

Die Tabelle, die einen Primärschlüssel als Fremdschlüssel zur Verfügung stellt, befindet sich in dem vorher erstellten Array an der 6. Position. Da die Zählung mit 0 beginnt, wird der Wert aus dem Array mit '**aTabelle(5)**' ermittelt.

```
inZaehler = 0
stFremdIDTab1Tab2 = "ID"
stFremdIDTab2Tab1 = "ID"
```

```
stNebentabelle = aTabelle(5)
```

Bevor die Auslesung des Arrays gestartet wird, müssen einige Standardwerte gesetzt werden. Dies sind der Zähler für das Array, in das die Werte der Nebentabelle geschrieben werden, der Standardprimärschlüssel, wenn nicht der Fremdschlüssel für eine zweite Tabelle benötigt wird und die Standardnebentabelle, die sich auf die Haupttabelle bezieht, bei Postleitzahl und Ort z.B. die Tabelle für die Postleitzahl.

Bei der Verknüpfung von zwei Feldern zur Anzeige in den Listenfeldern kann es ja, wie oben erwähnt, zu einer Verknüpfung über zwei Tabellen kommen. Für die Darstellung von Postleitzahl und Ort lautet hier die Abfrage

```
SELECT "Postleitzahl"."Postleitzahl" || ' > ' || "Ort"."Ort" FROM "Postleitzahl", "Ort" WHERE  
"Postleitzahl"."Ort_ID" = "Ort"."ID"
```

Die Tabelle, die sich auf das erste Feld bezieht (Postleitzahl), ist mit der zweiten Tabelle über einen Fremdschlüssel verbunden. Lediglich die Information der beiden Tabellen und der Felder "Postleitzahl" und "Ort" wurde dem Makro mitgegeben. Die Primärschlüssel sind standardmäßig in dem Beispiel mit der Bezeichnung "ID" versehen. Der Fremdschlüssel von "Ort" in "Postleitzahl" muss also über das Makro ermittelt werden.

Genauso muss über das Makro jede andere Tabelle ermittelt werden, mit der die Inhalte des Listenfeldes über Fremdschlüssel in Verbindung stehen.

```
oAbfrageergebnis = oSQL_Anweisung.executeQuery(stSql)  
IF NOT ISNULL(oAbfrageergebnis) THEN  
  WHILE oAbfrageergebnis.next  
    ReDim Preserve aFremdTabellen(inZaehler, 0 to 1)
```

Das Array muss jedes Mal neu Dimensioniert werden. Damit die alten Inhalte erhalten bleiben erfolgt über (Preserve) eine Sicherung des vorherigen Inhaltes.

```
aFremdTabellen(inZaehler, 0) = oAbfrageergebnis.getString(1)
```

Auslesen des ersten Feldes mit dem Namen der Tabelle, die den Fremdschlüssel enthält. Ergebnis für die Tabelle "Postleitzahl" ist hier die Tabelle "Adresse".

```
aFremdTabellen(inZaehler, 1) = oAbfrageergebnis.getString(2)
```

Auslesen des zweiten Feldes mit der Bezeichnung des Fremdschlüsselfeldes. Ergebnis für die Tabelle "Postleitzahl" ist hier das Feld "Postleitzahl_ID" in der Tabelle "Adresse".

Für den Fall, dass dem Aufruf der Prozedur auch der Name einer zweiten Tabelle mitgegeben wurde erfolgt die folgende Schleife. Nur wenn der Name der zweiten Tabelle als Fremdschlüsseltabelle für die erste Tabelle auftaucht erfolgt hier eine Änderung der Standardeinträge. In unserem Fall kommt dies nicht vor, da die Tabelle "Ort" keinen Fremdschlüssel der Tabelle "Postleitzahl" enthält. Der Standardeintrag für die Nebentabelle bleibt also bei "Postleitzahl"; schließlich ist die Kombination von Postleitzahl und Ort eine Grundlage für die Adressentabelle, die einen Fremdschlüssel zu der Tabelle "Postleitzahl" enthält.

```
IF UBound(aTabelle()) = 8 THEN  
  IF aTabelle(8) = aFremdTabellen(inZaehler, 0) THEN  
    stFremdIDTab2Tab1 = aFremdTabellen(inZaehler, 1)  
    stNebentabelle = aTabelle(8)  
  END IF  
END IF  
inZaehler = inZaehler + 1
```

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

```
WEND  
END IF
```

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

```
IF UBound(aTabelle()) = 8 THEN
```

Der Ablauf ist identisch. Nur wird in der Schleife jetzt gesucht, ob vielleicht der erste Tabellename als Fremdschlüssel-Tabellename auftaucht. Das ist hier der Fall: Die Tabelle "Postleitzahl" enthält den Fremdschlüssel "Ort_ID" aus der Tabelle "Ort". Dieser Fremdschlüssel wird also jetzt der Variablen «stFremdIDTab1Tab2» zugewiesen, so dass die Beziehung der Tabellen untereinander definiert werden kann.

```
IF aTabelle(5) = aFremdTabellen2(inZaehler,0) THEN
    stFremdIDTab1Tab2 = aFremdTabellen2(inZaehler,1)
END IF
```

Nach einigen weiteren Einstellungen zur korrekten Rückkehr nach Aufruf des Dialogs in die entsprechenden Formulare (Ermittlung der Zeilennummer des Formulars, damit nach einem Neueinlesen auf die Zeilennummer wieder gesprungen werden kann) startet die Schleife, die den Dialog gegebenenfalls wieder neu erstellt, wenn die erste Aktion erfolgt ist, der Dialog aber für weitere Aktionen offen gehalten werden soll. Die Einstellung zur Wiederholung erfolgt über das entsprechende Markierfeld.

DO

Bevor der Dialog gestartet wird, wird erst einmal der Inhalt der Listenfelder ermittelt. Dabei muss berücksichtigt werden, ob die Listenfelder zwei Tabellenfelder darstellen und eventuell sogar einen Bezug zu zwei Tabellen haben.

```
IF UBound(aTabelle()) = 6 THEN
```

Das Listenfeld bezieht sich nur auf eine Tabelle und ein Feld, da das Array bei dem Tabellenfeld1 der Nebentabelle endet.

```
    stSql = "SELECT "" + aTabelle(6) + "" FROM "" + aTabelle(5) + "" ORDER
BY "" + aTabelle(6) + ""
    ELSEIF UBound(aTabelle()) = 7 THEN
```

Das Listenfeld bezieht sich auf zwei Tabellenfelder, aber nur auf eine Tabelle, da das Array bei dem Tabellenfeld2 der Nebentabelle endet.

```
    stSql = "SELECT "" + aTabelle(6) + ""||' > '|"" + aTabelle(7) + ""
FROM "" + aTabelle(5) + "" ORDER BY "" + aTabelle(6) + ""
    ELSE
```

Das Listenfeld hat zwei Tabellenfelder und zwei Tabellen als Grundlage. Diese Abfrage trifft also auf das Beispiel mit der Postleitzahl und den Ort zu.

```
    stSql = "SELECT "" + aTabelle(5) + ""." + aTabelle(6) + ""||' > '|""
+ aTabelle(8) + ""." + aTabelle(7) + "" FROM "" + aTabelle(5) + "", "" +
aTabelle(8) + "" WHERE "" + aTabelle(8) + ""." + stFremdIDTab2Tab1 + "" = "" +
aTabelle(5) + ""." + stFremdIDTab1Tab2 + "" ORDER BY "" + aTabelle(6) + ""
    END IF
```

Hier erfolgt die erste Auswertung zur Ermittlung von Fremdschlüsseln. Die Variablen «stFremdIDTab2Tab1» und «stFremdIDTab1Tab2» starten mit dem Wert "ID". Für «stFremdIDTab1Tab2» wurde in der Auswertung der vorhergehenden Abfrage ein anderer Wert ermittelt, nämlich der Wert "Ort_ID". Damit ergibt die vorherige Abfragekonstruktion genau den Inhalt, der weiter oben bereits für Postleitzahl und Ort formuliert wurde – lediglich erweitert um die Sortierung.

Jetzt muss der Kontakt zu den Listenfeldern erstellt werden, damit diese mit dem Inhalt der Abfragen bestückt werden. Diese Listenfelder existieren noch nicht, da noch gar kein Dialog existiert. Dieser Dialog wird mit den folgenden Zeilen erst einmal im Speicher erstellt, bevor er tatsächlich auf dem Bildschirm ausgeführt wird.

```
DialogLibraries.LoadLibrary("Standard")
oDlg = CreateUnoDialog(DialogLibraries.Standard.Dialog_Tabellenbereinigung)
```

Anschließend werden Einstellungen für die Felder, die der Dialog enthält, ausgeführt. Hier als Beispiel das Auswahllistenfeld, das mit dem Ergebnis der obigen Abfrage bestückt wird:

```
oCtlList1 = oDlg.GetControl("ListBox1")
oCtlList1.addItem(aInhalt(),0)
```

Der Zugriff auf die Felder des Dialogs erfolgt über **'GetControl'** sowie die entsprechende Bezeichnung. Bei Dialogen ist es nicht möglich, für zwei Felder die gleichen Bezeichnungen zu verwenden, da sonst eine Auswertung des Dialoges problematisch wäre.

Das Listenfeld wird mit den Inhalten aus der Abfrage, die in dem Array «aInhalt()» gespeichert wurden, ausgestattet. Das Listenfeld enthält nur die darzustellenden Inhalte als ein Feld, wird also nur in der Position '0' bestückt.

Nachdem alle Felder mit den gewünschten Inhalten versorgt wurden wird der Dialog gestartet.

```
Select Case oDlg.Execute()  
Case 1 'Case 1 bedeutet die Betätigung des Buttons "OK"  
Case 0 'Wenn Button "Abbrechen"  
    inWiederholung = 0  
End Select  
LOOP WHILE inWiederholung = 1
```

Der Dialog wird so lange durchgeführt, wie der Wert für «inWiederholung» auf 1 steht. Diese Setzung erfolgt mit dem entsprechenden Markierfeld.

Hier der Inhalt nach Betätigung des Buttons «OK» im Kurzüberblick:

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

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

```
stText = oCtlText.Text ' Den Wert des Feldes auslesen.
```

Ist das Textfeld nicht leer, so wird nur der Eintrag im Textfeld erledigt. Weder das Listenfeld für eine andere Zuweisung noch das Markierfeld für eine Löschung aller Daten ohne Bezug werden berücksichtigt. Dies wird auch dadurch verdeutlicht, dass bei Texteingabe die anderen Felder inaktiv geschaltet werden.

```
IF stText <> "" THEN
```

Ist das Textfeld nicht leer, dann wird der neue Wertes anstelle des alten Wertes mit Hilfe des vorher ausgelesenen ID-Feldes in die Tabelle geschrieben. Dabei werden wieder zwei Einträge ermöglicht, wie dies auch in dem Listenfeld geschieht. Das Trennzeichen ist «>>». Bei Zwei Einträgen in verschiedenen Tabellen müssen auch entsprechend zwei UPDATE-Kommandos gestartet werden, die hier gleichzeitig erstellt werden und, durch ein Semikolon getrennt, weitergeleitet werden.

```
ELSEIF oCtlList2.getSelectedItem() <> "" THEN
```

Wenn das Textfeld leer ist und das Listenfeld 2 einen Wert aufweist muss der Wert des Listenfeldes 1 durch den Wert des Listenfeldes 2 ersetzt werden. Das bedeutet, dass alle Datensätze der Tabellen, in denen die Datensätze der Listenfelder Fremdschlüssel sind, überprüft und gegebenenfalls mit einem geänderten Fremdschlüssel beschrieben werden müssen.

```
stInhalt2 = oCtlList2.getSelectedItem()  
REM Den Wert der Listbox auslesen.  
REM ID für den Wert das Listenfeld ermitteln.
```

Der ID-Wert des zweiten Listenfeldes wird in der Variablen «inLB2» gespeichert. Auch dieses erfolgt wieder unterschiedlich, je nachdem, ob ein oder zwei Felder in dem Listenfeld enthalten sind, sowie eine oder zwei Tabellen Ursprungstabellen des Listenfeldinhaltes sind.

Der Ersetzungsprozess erfolgt danach, welche Tabelle als die Tabelle definiert wurde, die für die Haupttabelle den Fremdschlüssel darstellt. Für das oben erwähnte Beispiel ist dies die Tabelle "Postleitzahl", da die "Postleitzahl_ID" der Fremdschlüssel ist, der durch Listenfeld 1 und Listenfeld 2 wiedergegeben wird.

```
IF stNebentabelle = aTabelle(5) THEN  
    FOR i = LBound(aFremdTabellen()) TO UBound(aFremdTabellen())
```

Ersetzen des alten ID-Wertes durch den neuen ID-Wert. Problematisch ist dies bei n:m-Beziehungen, da dann der gleiche Wert doppelt zugeordnet werden kann. Dies kann erwünscht sein, muss

aber vermieden werden, wenn der Fremdschlüssel hier Teil des Primärschlüssels ist. So darf in der Tabelle "rel_Medien_Verfasser" ein Medium nicht zweimal den gleichen Verfasser haben, da der Primärschlüssel aus der "Medien_ID" und der "Verfasser_ID" gebildet wird. In der Abfrage werden alle Schlüsselfelder untersucht, die zusammen die Eigenschaft UNIQUE haben oder als Fremdschlüssel mit der Eigenschaft 'UNIQUE' über einen Index definiert wurden.

Sollte also der Fremdschlüssel die Eigenschaft 'UNIQUE' haben und bereits mit der gewünschten zukünftigen «inLB2» dort vertreten sein, so kann der Schlüssel nicht ersetzt werden.

```
stSql = "SELECT ""COLUMN_NAME"" FROM ""INFORMATION_SCHEMA"". ""SYSTEM_INDEXINFO""
WHERE ""TABLE_NAME"" = '' + aFremdTabellen(i,0) + '' AND ""NON_UNIQUE"" = False AND
""INDEX_NAME"" = (SELECT ""INDEX_NAME"" FROM
""INFORMATION_SCHEMA"". ""SYSTEM_INDEXINFO"" WHERE ""TABLE_NAME"" = '' +
aFremdTabellen(i,0) + '' AND ""COLUMN_NAME"" = '' + aFremdTabellen(i,1) + '' )"
```

Mit ' "NON_UNIQUE" = False ' werden die Spaltennamen angegeben, die 'UNIQUE' sind. Allerdings werden nicht alle Spaltennamen benötigt sondern nur die, die gemeinsam mit dem Fremdschlüsselfeld einen Index bilden. Dies ermittelt der 'Subselect' mit dem gleichen Tabellennamen (der den Fremdschlüssel enthält) und dem Namen des Fremdschlüsselfeldes.

Wenn jetzt der Fremdschlüssel in der Ergebnismenge vorhanden ist, dann darf der Schlüsselwert nur dann ersetzt werden, wenn gleichzeitig andere Felder dazu benutzt werden, den entsprechenden Index als 'UNIQUE' zu definieren. Hierzu muss beim Ersetzen darauf geachtet werden, dass die Einzigartigkeit der Indexkombination nicht verletzt wird.

```
IF aFremdTabellen(i,1) = stFeldbezeichnung THEN
    inUnique = 1
ELSE
    ReDim Preserve aSpalten(inZaehler)
    aSpalten(inZaehler) = oAbfrageergebnis.getString(1)
    inZaehler = inZaehler + 1
END IF
```

Alle Spaltennamen, die neben dem bereits bekannten Spaltennamen des Fremdschlüsselfeldes als Index mit der Eigenschaft UNIQUE auftauchen, werden in einem Array abgespeichert. Da der Spaltenname des Fremdschlüsselfeldes auch zu der Gruppe gehört wird durch ihn gekennzeichnet, dass die Einzigartigkeit bei der Datenänderung zu berücksichtigen ist.

```
IF inUnique = 1 THEN
    stSql = "UPDATE "" + aFremdTabellen(i,0) + "" AS ""a"" SET "" +
aFremdTabellen(i,1) + ""="" + inLB2 + "" WHERE "" + aFremdTabellen(i,1) + ""="" +
inLB1 + "" AND ( SELECT COUNT(*) FROM "" + aFremdTabellen(i,0) + "" WHERE "" +
aFremdTabellen(i,1) + ""="" + inLB2 + "" )"
    IF inZaehler > 0 THEN
        stFeldgruppe = Join(aSpalten(), ""|| ||"" )
```

Gibt es mehrere Felder, die neben dem Fremdschlüsselfeld gemeinsam einen 'UNIQUE'-Index bilden, so werden die hier für eine SQL-Gruppierung zusammengeführt. Ansonsten erscheint als «stFeldgruppe» nur «aSpalten(0)».

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

Die SQL-Teilstücke werden für eine korrelierte Unterabfrage zusammengefügt.

```
NEXT
stSql = Left(stSql, Len(stSql) - 1)
```

Die vorher erstellte Abfrage endet mit einer Klammer. Jetzt sollen noch Inhalte zu der Unterabfrage hinzugefügt werden. Also muss die Schließung wieder aufgehoben werden. Anschließend wird die Abfrage durch die zusätzlich ermittelten Bedingungen ergänzt.

```
stSql = stSql + stFeldbezeichnung + "GROUP BY ("" + stFeldgruppe + "" ) < 1"
END IF
```

Wenn die Feldbezeichnung des Fremdschlüssels nichts mit dem Primärschlüssel oder einem 'UNIQUE'-Index zu tun hat, dann kann ohne weiteres auch ein Inhalt doppelt erscheinen

```

ELSE
    stSql = "UPDATE "" + aFremdTabellen(i,0) + "" SET "" + aFremdTabellen(i,1) +
""=' + inLB2 + '' WHERE "" + aFremdTabellen(i,1) + ""=' + inLB1 + ''
END IF
oSQL_Anweisung.executeQuery(stSql)
NEXT

```

Das Update wird so lange durchgeführt, wie unterschiedliche Verbindungen zu anderen Tabellen mit vorkommen, d.h. Die aktuelle Tabelle einen Fremdschlüssel in anderen Tabellen liegen hat. Dies ist z.B. Bei der Tabelle "Ort" zweimal der Fall: in der Tabelle "Medien" und in der Tabelle "Postleitzahl".

Anschließend kann der alte Wert aus dem Listenfeld 1 gelöscht werden, weil er keine Verbindung mehr zu anderen Tabellen hat.

```

stSql = "DELETE FROM "" + aTabelle(5) + "" WHERE ""ID""=' + inLB1 + ''
oSQL_Anweisung.executeQuery(stSql)

```

Das gleiche Verfahren muss jetzt auch für eine eventuelle zweite Tabelle durchgeführt werden, aus der die Listenfelder gespeist werden. In unserem Beispiel ist die erste Tabelle die Tabelle "Postleitzahl", die zweite Tabelle die Tabelle "Ort".

Wenn das Textfeld leer ist und das Listenfeld 2 ebenfalls nichts enthält wird nachgesehen, ob eventuell das Markierfeld darauf hindeutet, dass alle überflüssigen Einträge zu löschen sind. Dies ist für die Einträge der Fall, die nicht mit anderen Tabellen über einen Fremdschlüssel verbunden sind.

```

ELSEIF oCtlCheck1.State = 1 THEN
    stBedingung = ""
    IF stNebentabelle = aTabelle(5) THEN
        FOR i = LBound(aFremdTabellen()) TO UBound(aFremdTabellen())
            stBedingung = stBedingung + ""ID"" NOT IN (SELECT "" +
aFremdTabellen(i,1) + "" FROM "" + aFremdTabellen(i,0) + "") AND "
        NEXT
    ELSE
        FOR i = LBound(aFremdTabellen2()) TO UBound(aFremdTabellen2())
            stBedingung = stBedingung + ""ID"" NOT IN (SELECT "" +
aFremdTabellen2(i,1) + "" FROM "" + aFremdTabellen2(i,0) + "") AND "
        NEXT
    END IF

```

Das letzte «AND» muss abgeschnitten werden, da sonst die Löschanweisung mit einem «AND» enden würde.

```

stBedingung = Left(stBedingung, Len(stBedingung) - 4) '
stSql = "DELETE FROM "" + stNebentabelle + "" WHERE " + stBedingung + ""
oSQL_Anweisung.executeQuery(stSql)

```

Da nun schon einmal die Tabelle bereinigt wurde kann auch gleich der Tabellenindex überprüft und gegebenenfalls nach unten korrigiert werden. Siehe hierzu die in dem vorhergehenden Kapitel [Tabellenindex heruntersetzen bei Autowert-Feldern](#) erwähnte Prozedur.

```

Tabellenindex_runter(stNebentabelle)

```

Anschließend wird noch gegebenenfalls das Listenfeld des Formulars, aus dem der Tabellenbereinigungsdialog aufgerufen wurde, auf den neuesten Stand gebracht. Unter Umständen ist das gesamte Formular neu einzulesen. Hierzu wurde zu Beginn der Prozedur der aktuelle Datensatz ermittelt, so dass nach einem Auffrischen des Formulars der aktuelle Datensatz auch wieder eingestellt werden kann.

```

oDlg.endExecute() 'Dialog beenden ...
oDlg.Dispose() '... und aus dem Speicher entfernen
END SUB

```

Dialoge werden mit «endExecute()» beendet und mit «Dispose()» komplett aus dem Speicher entfernt.