LibreOffice
The Document Foundation

Erste Schritte

Kapitel 13
Einführung in Makros

Immer wiederkehrende Schritte automatisieren

Copyright

Mitwirkende/Autoren

Jochen Schiffers

Gerhard Weydt

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 04.07.2017 Basierend auf der LibreOffice Version 5.2.

Inhalt

Einleitung

Dieses Kapitel soll Ihnen einen ersten Überblick über die Möglichkeiten geben, die eine Erweiterung der Standardfunktionalität durch – selbstgeschriebene oder von anderswo erhaltene – Makros bietet. In der notwendigen Kürze kann sie natürlich den Raum der Möglichkeiten nur erahnen lassen und auch keine auch nur halbwegs vollständige Anleitung zum Programmieren in LibreOffice geben. Dies bleibt Ihrer eigenen Initiative überlassen, zu der Sie am Ende des Kapitels Hinweise zu weiteren Schritten finden.

Makros in LibreOffice

Makros sind Befehlsfolgen, die abgespeichert sind und die Sie somit immer wieder ausführen können. Dadurch sparen Sie sich erhebliche Tipp-, Klick- und Denkarbeit, wiederkehrende Aufgaben können Sie so deutlich schneller ausführen.

Wenn man dann noch die Möglichkeiten von Dialogen und Formularen und Datenbanken hinzunimmt, die LibreOffice anbietet (siehe dazu vor allem das Kapitel „Base“ in diesem Handbuch), dann lassen sich ganze eigene Anwendungen erstellen, mit denen man z.B. durchaus ein kleines Geschäft verwalten kann.

Makros können Sie selber erstellen oder von Bekannten oder aus dem Internet erhalten und lediglich in Ihrem System speichern. Auch die Extensions von LibreOffice (siehe dazu das Kapitel „Anpassen von LibreOffice“ in diesem Handbuch) bestehen zu einem Großteil aus Makros. Außerdem enthält die Installation von LibreOffice bereits eine Sammlung von Makros, im Wesentlichen Hilfsfunktionen bereitstellen, die Sie für eigene Programme verwenden können.

Vorsicht 	Wenn Sie Makros von jemand anderem erhalten, sollten Sie sie vor dem Einsatz prüfen, wenn Sie nicht sicher sind, dass Sie der Quelle vertrauen können. Makrocode ist reiner Text, er kann also keine Viren o.ä., wohl aber Fehler, die unangenehme Folgen auslösen können, und im Extremfall auch böswilligen Code enthalten. Sie sollten also die Funktionsweise eines Makros verstanden haben, bevor Sie es einsetzen.

Hinweise bzgl. Makro-Beispiele in diesem Handbuch

In dieser ersten Einführung kann keine umfassende Beschreibung des Programmierens in LibreOffice gegeben werden. Es sollen nur an einigen aufeinander aufbauenden Beispielen die wesentlichen Aspekte dargestellt werden, um Ihnen einen Eindruck von den Möglichkeiten zu verschaffen, die die Makros bieten.

Als Thema für die Beispiele wird der Übergang von Kontonummer und Bankleitzahl zur IBAN in Deutschland verwendet, ein Thema, mit dem praktisch jeder konfrontiert war. Daran lassen sich diverse Möglichkeiten des Einsatzes von Makros demonstrieren, die verständlich, jedoch nicht trivial sind. Sie sollen weder mit wieder einer neuen Variante eines „Hello World“-Programms gelangweilt werden noch mit Aufgabenstellungen, die sich auch ohne Programmieren lösen lassen. Denn vieles, was an wiederkehrenden, komplizierteren Aktionen benötigt wird, lässt sich ja bereits mit Standardmitteln lösen, z.B. der Mehrfacheinsatz eines Textes oder Zeichnungsobjekts mitsamt allen Formatierungen durch Kopieren in die Zwischenablage und Einfügen, die Formatübertragung oder das Suchen und Ersetzen.

Das erste Programm enthält also bereits einige Zeilen mehr als ein minimales lauffähiges Programm, ist dafür aber bereits interessant und zeigt einige typische Elemente, die Sie in jedem Makro finden werden. Sie werden sogar die Behebung eines einfach zu lokalisierenden Fehlers sehen.

Basic

Bei der Programmierung von Makros wird in LibreOffice meist eine Variante der interpretierenden Programmiersprache Basic verwendet. Interpretierend heißt, dass beim Ausführen eines Makros der Programmcode Anweisung für Anweisung in Rechnerbefehle umgesetzt und ausgeführt wird. Im Gegensatz dazu wird ein Programm, das in einer sogenannten kompilierenden Sprache erstellt wurde, in einem ersten Arbeitsschritt als Ganzes in maschinenlesbaren Code übersetzt, eventuell auch noch mit weiteren, bei dieser Übersetzung als notwendig erkannten Komponenten oder weiteren, getrennt erstellten Programmen verbunden und in dieser Form gespeichert; dieses ausführbare Programm wird dann bei Bedarf aufgerufen.

Kompilierte Programme sind in der Ausführung schneller, da alles, was sich schon vorab erledigen lässt, bereits geschehen ist und der ausführbare Code optimiert ist. Dieser Gewinn an Geschwindigkeit fällt jedoch angesichts der Leistungsfähigkeit heutiger Rechner bei fast allen Aufgabenstellungen, die man in LibreOffice und anderswo lösen möchte, nicht mehr ins Gewicht. Hingegen ist die Entwicklung mit Basic einfacher, weil man jede Änderung ohne Zwischenschritt sofort ausprobieren kann; außerdem ist Basic-Code ohne jede Änderung auch in einer LibreOffice-Installation einsetzbar, die ein anderes Betriebssystem verwendet.

Objekte von LibreOffice

Die Sprache Basic hat einen Vorrat von Befehlen und Funktionen, der gut überschaubar und daher sehr schnell zu beherrschen ist. Die ersten drei Beispiele von Makros in den folgenden Abschnitten arbeiten auch nur mit diesem Inventar. Man kommt allerdings ganz schnell – und das gilt nicht bloß für LibreOffice – zu dem Punkt, wo man nicht nur Werte berechnen und Texte verketten, sondern sich mit den Objekten des Systems, also z.B. Texten, Textteilen, Präsentationen oder grafischen Formen und deren Eigenschaften beschäftigen möchte. Hierzu muss man sich nun nicht mit der Form auskennen, wie solche Informationen, beispielsweise die Formatierung eines Zeichens, im Dokument physisch gespeichert werden. Es werden vielmehr für alle notwendigen Konstrukte wie z.B. Text, Absatz, Textteil, also eine zusammenhängende Folge von Zeichen mit gleichen Eigenschaften, aber auch eher abstrakte Gebilde wie der Controller, der alles weiß, was an der Oberfläche passiert, also wo der Cursor steht oder was Sie markiert haben, sogenannte Objekte des UNO-Modells (Universal Network Objects) zur Verfügung gestellt, auf die man von Basic aus zugreifen kann. Diese Objekte sind nicht auf Basic beschränkt, vielmehr bilden sie das Herz der Architektur von LibreOffice.

Ein Objekt enthält meist Dutzende von Eigenschaften (Attributen), die einfache sein können wie die Schriftfarbe eines Textabschnitts, aber auch komplexe, die selbst Objekte oder Ansammlungen von Objekten sind. So enthält dieses Dokument, das auch ein UNO-Objekt ist, als Attribute unter anderem den eigentlichen Text, die Sammlung aller Fußnoten, die Sammlung aller im Dokument enthaltenen Tabellen usw. Jedes der genannten Objekte enthält wiederum neben einfachen auch komplexe Attribute, die ebenso Objekte sind. Die Sammlung der Tabellen enthält eben alle Tabellen, jede Tabelle ihre Zellen, jede Zelle ihre Absätze und diese die Textabschnitte; irgendwann ist man bei elementaren Objekten angelangt, die ähnlich sind wie die die Datentypen von Basic oder Kombinationen von solchen, wie – als einfaches Beispiel – die Größe z.B. eines Rechtecks, die aus zwei Zahlenwerten für Höhe und Breite besteht.

Weiter enthält ein Objekt Methoden, deren Aufruf das Objekt veranlasst, etwas zu tun. Das können einfache Aktionen wie das Setzen eines Attributs sein, was meist genauso gut durch eine Zuweisung geschehen kann, aber auch aufwändigere wie z.B. das Drucken oder das Speichern eines Dokuments mit all den dazu notwendigen Parametern, wie Sie sie von den dabei aufgerufenen Dialogen kennen.

Diese Objekte machen das Programmieren einerseits wesentlich leichter, ja, sie ermöglichen eigentlich erst ein Programmieren mit vertretbarem Aufwand, weil sie sehr komplizierte Konstellationen in verdaubare Häppchen aufteilen und mächtige Funktionen bereitstellen. Andererseits erfordert es eine gewisse Zeit, sich mit den Objekten, die man für eine Aufgabenstellung benötigt,

vertraut zu machen. Wir werden anhand des vierten Beispiels („Beispiel 4: Massensetzung der IBAN in einer Calc-Tabelle“ auf Seite 24) sehen, wie das funktioniert.

Erstellen von Makros

Eine komplette Beschreibung der Syntax kann hier nicht gegeben werden, wo Sie dazu Informationen finden, ist in „Weitere Schritte“ auf Seite 36 beschrieben. Daher werden aufeinander aufbauende Beispiele verwendet, um die wesentlichen Aspekte darzustellen. Es wird dazu der Programmcode jeweils zunächst aufgelistet und dann erläutert.

Nach den ersten Beispielen (1 bis 3), die nur Basic-Code enthalten, folgen auch zwei (4 bis 5), die die Verwendung von Basic im Zusammenhang mit den Objekten von LibreOffice zeigen, die für den Zugriff auf Dokumente und ihre Komponenten genutzt werden können.

Da dies eine Einführung ist, wird versucht, bei den Erklärungen praktisch bei Null anzufangen. Wer schon einmal in irgendeiner Sprache programmiert hat, wird die folgenden Seiten sehr schnell überfliegen, weil ihm das meiste recht vertraut vorkommt und er die Abweichungen sofort einordnen kann.

Beispiel 1: IBAN berechnen

Aufgabenstellung

Im internationalen Zahlungsverkehr hat man sich auf die Verwendung einer allgemein gültigen Internationalen Bank-Identifikations-Nummer (IBAN) geeinigt). In Europa werden dadurch die bisherigen Begriffe Kontonummer und Bankleitzahl ersetzt. Der Aufbau der IBAN ist länderspezifisch unterschiedlich, die ersten Stellen sind jedoch überall gleich:

- die ersten zwei Zeichen bestehen aus dem Länderkürzel, z.B. „DE“ für Deutschland
- die dritte und vierte Stelle beinhaltet die Prüfziffern, die aus dem restlichen Inhalt der IBAN berechnet werden und die Tipp- und Übertragungsfehler aufdecken sollen

In Deutschland folgen darauf die achtstellige Bankleitzahl und die eventuell mit führenden Nullen auf 10 Stellen aufgefüllte Kontonummer.

Damit ist das Verfahren der Bildung der IBAN für Deutschland klar und einfach bis auf die Bildung der Prüfziffern, deren Berechnung allerdings in Wikipedia auffindbar ist.

Vorsicht 	Das im Folgenden dargestellte.Programm beinhaltet nur die Logik für den Standardfall – und das nur für Deutschland! Sonderfälle, z.B. wegen Unterkonten, Handhabung von historischen Altlasten usw. sind nicht abgedeckt. Deren Behandlung würde das Programm im vorliegenden Kontext undurchschaubar machen. Tatsächlich ist die Anzahl der Ausnahmen und Sonderfälle so groß, dass offiziell gültige IBANs nur von Banken mitgeteilt werden dürfen. Die vom Beispielprogramm gelieferten IBANs könnten aber als Vorschlag verwendet werden, der vom Empfänger bestätigt werden soll und in den meisten Fällen auch schon den richtigen Wert liefern wird.

Das erste Makro, mit einem kleinen Problem

Hier sehen Sie zunächst eine erste Version des Makros, die allerdings einen kleinen Fehler beinhaltet, der kein logischer Fehler ist, sondern mit den Eigenheiten des Programmierens zusammenhängt und sich keinesfalls auf Basic beschränkt. Dadurch ist aber der Code zunächst einmal kürzer und besser verständlich. Diesen Fehler werden wir, nachdem zunächst die einzelnen Zeilen des Makros erläutert wurden, gleich bei der Ausführung feststellen und bei der Gelegenheit ein

Beispiel sehen, welche Fehlermeldungen auftauchen können. Wie ein Makro gespeichert und ausgeführt wird, sehen Sie dann im Anschluss an die korrigierte Version des Makros in „Makros verwalten“ auf Seite 13.

Der Code ist bei diesem Beispiel so eingefärbt, wie Sie ihn beim Verwalten sehen (Schlüsselwörter blau, Variable und Funktionen grün, Konstanten rot und Kommentare grau), in späteren Beispielen wird die Färbung weggelassen.

Vorsicht

Der gezeigt Code führt zu einer Fehlermeldung, der korrekte Code folgt im nächsten Beispiel.

```
sub ermittleIBAN
REM vereinfachte Variante: nur für Deutschland, nur Standardfälle
REM abgedeckt.   Quelle: deutsche Wikipedia

dim blz as string, konto as string
dim ibanBasis as string, iban as string
dim rest as long

blz = "79876543"
konto = "0012345678"

ibanBasis = blz & konto & "131400"
   'die letzten Stellen ergeben sich aus den vorgeschriebenen Ziffern
   'für die Buchstaben D und E und "00" da, wo später die Prüfziffer
   'steht
rest = ibanBasis MOD 97       ' Rest modulo 97
iban = "DE" & Format(98 - rest, "00") & blz & konto

msgbox (iban, 64, "ermittelte IBAN")

end sub
```

Die meisten Schreibweisen im folgenden Programm sind jedem, der schon einmal eine Programmiersprache verwendet hat, sicher vertraut.

- Das Programm beginnt mit der Anweisung „sub“ (kurz für subroutine = (Unter)-Programm); die zweite Möglichkeit für den Programmbeginn sehen Sie im nächsten Abschnitt. Danach folgt der Name des Programms, den Sie selbst bestimmen können.
  Namen – für Programme wie für Variablen – sollten in Ihrem eigenen Interesse aussagekräftig sein.
- Danach folgt hier ein Kommentar, der an jeder beliebigen Stelle des Codes eingefügt werden kann – und aus Gründen der Dokumentation auch großzügig eingefügt werden sollte. Ein Kommentar kann sowohl mit dem Schlüsselwort „REM“ (für englisch remark = Anmerkung; die Großschreibung hat sich eingebürgert, Basic macht keinen Unterschied zwischen Groß-und Kleinbuchstaben) als auch mit einem einfachen Anführungszeichen begonnen werden. Ein Kommentar kann auch mitten in einer Zeile, nach einer Anweisung, beginnen, wie in der Zeile zu sehen ist, die mit „rest“ beginnt.
- Die Variablen, die wir benötigen – die beiden Ausgangsfelder sowie Zwischenwerte und das Endergebnis – werden mit dem Schlüsselwort „dim“ definiert. Hinter „as“ folgt der Typ der Variablen, hier sind das Zeichenketten („string“), die sehr lange Ketten von Zeichen, und große Ganzzahlwerte („long“), die Zahlen bis zu mehr als einer Million aufnehmen können. Die Variablennamen können Sie frei vergeben, aussagekräftige Namen sind von Vorteil.

Sie können mehrere Variable in einer Anweisung definieren oder auch für jede Variable eine neue Anweisung verwenden.

- Den beiden Feldern für Bankleitzahl und Konto, von denen wir ausgehen, werden die (fiktiven) Werte zugewiesen. Die Kontonummer wird 10-stellig vorausgesetzt, daher wurde sie vorne mit führenden Nullen ergänzt.
- Für die Berechnung der Prüfziffer ist laut Vorschrift hinter Bankleitzahl und 10-stelliger Kontonummer das in Zahlen umgesetzte Länderkürzel, hier für Deutschland („1314 für „DE“) und „00“ als Platzhalter für die Prüfziffer zu setzen. Die Variable, in der wir diese Zeichenkette speichern, heißt, wie Sie in der nächsten Zeile sehen, „ibanBasis“ (für Basic ist die Unterscheidung zwischen Groß- und Kleinbuchstaben irrelevant; für den Benutzer gliedert ein Großbuchstabe ein längeres, zusammengesetztes Wort aber optisch sehr hilfreich). Die Zusammensetzung aus den Teilfeldern erfolgt mit dem Verkettungsoperator „&“: die Zeichenketten werden einfach hintereinander angestückelt.
- Nun folgt mit der Anweisung
  rest = ibanBasis mod 97
  die Berechnung der IBAN. Der verwendete Operator *Mod* bedeutet, dass der Rest bei der Teilung von ibanBasis durch 97 zu ermitteln ist (dieser Operator heißt „modulo“ und ist eine in der Mathematik übliche, die LibreOffice-Basic deshalb schon bereitstellt, so dass Sie sie nicht mehr eigens programmieren müssen, wenn Sie sie benötigen). Das Ergebnis ist eine Ganzzahl zwischen 0 und 96.
  Beachten Sie, dass wir die Variable ibanBasis vom Typ string zum Berechnen verwenden; Basic versucht in solchen Fällen eine Typumwandlung vor der Berechnung. Das Wort „versucht“ steht hier bewusst, wie Sie gleich sehen werden.
- Zeile „iban = ...“: Laut Vorschrift ist nun der ermittelte Rest von 98 abzuziehen. Die Zahl kann eine oder zwei Stellen haben, bei einer Stelle ist eine Null voranzustellen; das erledigt elegant die Funktion „Format“, die viel mehr als nur das kann, deren hier angegebener Parameter “00“ aber einfach besagt, das das Ergebnis – eine Zeichenkette, wie wir sie ja jetzt wieder brauchen – auf jeden Fall zwei Stellen haben soll und dass bei Bedarf links mit Nullen gefüllt werden soll.
  Sie sehen, dass man, anstatt jeden einzelnen Schritt mit einer neuen Variablen zu beginnen, durchaus auch kompliziertere Ausdrücke als Parameter in einer Funktion – in diesem Fall eine einfache Differenz, aber das kann weit komplexer sein – verwenden kann. Man muss hier zwischen Bequemlichkeit und Lesbarkeit abwägen.
  Das Vorgehen in diesen Zeilen illustriert auch die häufig vorkommende Situation, dass man in der Programmierung zwischen verschiedenen Datentypen hin- und herwechseln muss, hier mit der Funktion Format.
- In der genannten Zeile wird nun aus den Bestandteilen:
  - der für Deutschland fixen Zeichenkette „DE“
  - den gerade ermittelten Prüfziffern
  - der Bankleitzahl
  - und dem Konto

  die ermittelte IBAN zusammengesetzt. Der Operator „&“ wurde bereits weiter oben verwendet.
- Zum Schluss wird die ermittelte IBAN am Bildschirm ausgegeben. Dazu wird die von Basic bereitgestellte Funktion „msgbox“ (Messagebox) benutzt, die die Ausgabe ansprechend aufbereitet:
  - Im ersten Parameter wird der auszugebende Text benannt

- der zweite Parameter beschreibt die Darstellung der Ausgabe, welche Schaltflächen sichtbar sind und welches Symbol (hier als Beispiel das Zeichen für „Information“) angezeigt wird.
- als dritter Parameter wird die Überschrift mitgegeben

- mit „end sub“ wird das Programm beendet

Wenn Sie dieses Programm ausführen, erscheint nun, wie angekündigt die Fehlermeldung (Abbildung 1)

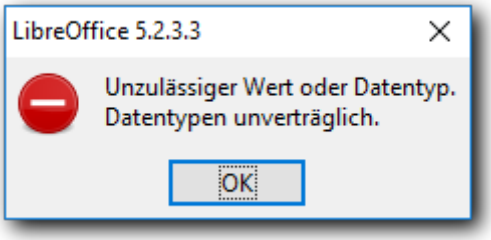

Abbildung 1: Fehlermeldung bei der Ausführung des Makros

und die Zeile, in der der Fehler auftritt, ist markiert (Abbildung 2):

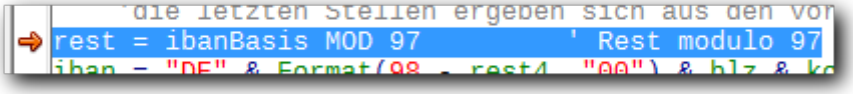

Abbildung 2: markierte Zeile bei Fehler

Finden der Fehlerursache

Es wurde schon gesagt, dass LibreOffice in dieser Anweisung eine Umwandlung der Stringvariable in eine Zahl versucht, bevor die Berechnung durchgeführt wird. Dabei entsteht aber, wie die Fehlermeldung besagt, offenbar ein Problem, das wir aber noch nicht genau einordnen können.

Wir ändern daher nun das Programm ab, indem wir die fehlerhafte Zeile durch

```
ibanBasisLong = ibanBasis
rest = ibanBasisLong MOD 97
```

ersetzen, sodass wir genauer steuern können, wie die Umwandlung geschieht. Die neue Variable deklarieren wir dazu in einer neuen dim-Anweisung:

```
dim ibanBasisLong as long
```

Ohne diese dim-Anweisung wäre nichts gewonnen, die Zuweisung würde einfach die neue Variable wieder als string ansehen, und wir wären genauso weit wie vorher. Basic bestimmt nämlich für neue Variablen, die nicht deklariert sind, den Typ aus dem Ergebnis der rechten Seite der Anweisung. Das ist bequem, weil man sich Tipparbeit spart, kann aber bei der Ausführung, wie wir hier sehen, zu Fehlern führen, die Sie manchmal auch gar nicht wie hier sofort als Fehler gemeldet bekommen, weil der ungewollt ermittelte Typ auch weiterverarbeitet werden kann.

Tipp	Deklarieren Sie immer alle Variablen, dann sind Sie sicher, dass der richtige Typ verwendet wird. Verwenden Sie am besten am Beginn jedes Moduls (s. „Erstellen eines Moduls“ auf Seite 16) die Anweisung „Option Explicit“, dann weist Sie LibreOffice darauf hin, dass Variablen nicht deklariert sind.

Wenn wir das so geändert Makro ausführen, erhalten wir zwar wieder eine Meldung (Abbildung 3), aber die bringt uns weiter:

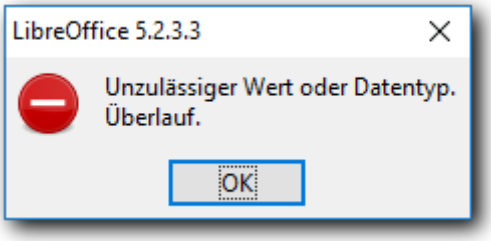

Abbildung 3: Fehlermeldung aus geändertem Makro

Das Wort „Überlauf“ besagt, dass der Wert, der der Variablen ibanBasisLong zu gewiesen wird, zu groß ist. Ganzzahlwerte in Basic können nämlich maximal nur 10 Dezimalstellen haben, wobei die vorderste Stelle nicht einmal voll bis zur Ziffer 9 ausgenutzt werden kann. Das liegt an der internen Speicherung als Binärzahlen mit 32 Bit. Eine größere Ganzzahl gibt es in Basic nicht, und diese Einschränkung findet sich in den meisten Programmiersprachen genauso.

In unserem Fall lässt sich das Problem aber sehr leicht beheben, wie Sie im nächsten Abschnitt sehen.

Beispiel 2: Das korrigierte Makro

Hier sehen Sie zunächst das korrigierte Makro. Zur Berechnung des Rests bei der Division durch 97 für diese große Zahl entsinnen wir uns der schriftlichen Teilung, wie wir sie aus dem Schulunterricht kennen.

```
sub ermittleIBAN
REM vereinfachte Variante: nur für Deutschland, nur Standardfälle
REM abgedeckt.   Quelle: deutsche Wikipedia

dim blz as string, konto as string
dim ibanBasis as string, iban as string
dim rest as long

blz = "79876543"
konto = "0012345678"

ibanBasis = blz & konto & "131400"
   'die letzten Stellen ergeben sich aus den vorgeschriebenen Ziffern
   'für die Buchstaben D und E und "00" da, wo später die Prüfziffer
   'steht
rest = Mid(ibanBasis,1,1)
for i = 2 to 24
   rest = (rest * 10 + Mid(ibanBasis,i,1)) MOD 97
next
iban = "DE" & Format(98 - rest, "00") & blz & konto

msgbox (iban, 64, "ermittelte IBAN")

end sub
```

Statt der fehlerhaften Zeile „rest = ibanBasis mod 97“ wird das Verfahren der schriftlichen Teilung programmiert. Da wir nur an dem Rest interessiert sind und nicht am Ergebnis der Division und da bei diesem Verfahren auch bei jedem Schritt der Rest in die weitere Berechnung eingeht, können wir uns die eigentliche Division und die daraus erst folgende Berechnung des Rests sparen und weiterhin mit der Funktion MOD arbeiten, die uns den Rest direkt liefert.

In einer Schleife holen wir uns jeweils die nächste Stelle der großen Zahl, die ja noch in einer Zeichenkette gespeichert ist, und hängen sie an den zuvor ermittelten Rest an.

- Vor Beginn der Schleife setzen wir als Anfangswert, mit dem Berechnung beginnt, die erste Ziffer der Zahl, die mit der Funktion *Mid* ermittelt wird. Diese Funktion schneidet aus der Zeichenkette, die an erster Stelle in der Klammer genannt ist (erster Parameter) eine Teilkette heraus, die an der Stelle des 2. Parameters beginnt (im ersten Fall also ab Stelle 1, dem Beginn) und die Länge hat, die der dritte Parameter angibt. Wir verwenden in diesem Programm nur die Länge 1, weil wir jeweils eine Ziffer ausschneiden wollen.
  Wir verwenden als Variable hier bereits rest, obwohl der Wert hier noch nicht als Rest einer Division ermittelt wurde, aber in den Schritten der Schleife, die dann folgen, ist der Ausgangswert immer der Rest einer Division, damit ist die eingesetzte Formel in allen Schritten einheitlich. Der Name einer Variablen hat ja für das Programm keine Bedeutung, sondern nur für den Leser.
- Nun beginnt die Schleife, die über alle Stellen der Zahl läuft außer der ersten, denn die haben wir ja schon behandelt. Es gibt in Basic mehrere Arten von Schleifen, die gebräuchlichste ist die hier verwendete *for*-Schleife, die über einen Index *i* gesteuert wird, der von einem Anfangswert, der hinter dem Gleichheitszeichen steht, bis zu einem Endwert, der hinter *to* genannt ist läuft, im einfachsten Fall wie hier, in Schritten von 1, das heißt, vor jedem Durchlauf der Schleife wird *i* um 1 erhöht; andere Schrittweiten müssen explizit angegeben werden.
- Zur *for*-Anweisung gehört weiter unten die *next*-Anweisung, die die Schleife abschließt: alle Anweisungen dazwischen werden für jedes *i* der durchgeführten Schritte ausgeführt.
- In jedem Schritt wird nun der Rest durch Anhängen der nächsten Ziffer ergänzt. In Zahlen umgesetzt bedeutet diese für Zeichen formulierte Vorschrift, dass der (mathematische) Rest mit 10 multipliziert wird, weil er um eine Stelle nach links rückt. Die nächste Ziffer wird wieder mit der Funktion *Mid* ermittelt, beachten Sie, dass dabei die Stelle, die ausgeschnitten wird, durch den Index *i* festgelegt wird.
- Für die so ermittelte Zahl wird jeweils der Rest modulo 97, also bei der Teilung durch 97, errechnet, das Ergebnis wird wieder in die Variable *rest* gestellt, damit es beim nächsten Schleifendurchlauf in gleicher Weise verarbeitet werden kann.
  Die Anweisung zeigt eine Eigenheit der Programmiersprachen, die völlige Neulinge zuerst vielleicht verwirrt, an die man sich aber in kürzester Zeit gewöhnt: Trotz des Gleichheitszeichens ist die Zeile keine Gleichung, sondern eine Zuweisung, der Variablenname *rest* links und rechts des Gleichheitszeichens bedeutet nicht das identische Objekt, sondern den Inhalt des gleichen Speicherplatzes zu unterschiedlichen Zeiten: rechts zu Beginn der Ausführung, als Eingabewert in eine Berechnung, deren Wert am Ende der Berechnung dann in den Speicherplatz, der links des Gleichheitszeichens benannt wird geschrieben wird, der nur in diesem Fall aus praktischen Gründen der gleiche ist wie eines der Ausgangsfelder.
  Die Einrückung der Anweisung(en) in der Schleife ist eine übliche Konvention, die die Lesbarkeit des Programms verbessert, indem die Anweisungen, die zur Schleife gehören, zusammengefasst und kenntlich gemacht werden.

Noch eleganter wird das Programm, wenn wir erkennen, dass die Sonderbehandlung der ersten Stelle nicht nötig ist: wenn wir mit 0 als Startwert beginnen und die Anweisung in der Schleife ausführen, ergibt sich als Rest der Teilung von 0 * 10 + erste Stelle durch 97 genau die erste Stelle; wir können also die Schleife von 1 bis 24 laufen lassen. Wenn wir noch ausnutzen, dass eine numerische Variable in Basic als Anfangswert vor der erstmaligen Verwendung immer den Wert 0 enthält, dann können wir uns die Zuweisung vor der Schleife ersparen.
Das Programm ist dann knapper und wird so auch in „Beispiel 3: IBAN ermitteln mit Prüfungen, Unterprogrammaufruf“ auf Seite 20) verwendet, aber für das Verständnis der Umsetzung der schriftlichen Division ist die hier gezeigte Variante wohl nützlicher.

Wenn dieses Programm ausgeführt wird, erscheint als Ergebnis das folgende Fenster mit der korrekten IBAN (Abbildung 4):

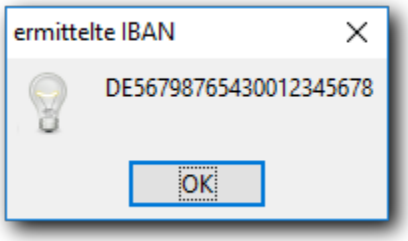

Abbildung 4: Ausgabe des Programms "ermittleIBAN"

Dieses Beispiel beinhaltet schon die meisten wesentlichen Bestandteile eines in (LibreOffice-) Basic geschriebenen Programms. Es gibt noch eine einige weitere Datentypen, einige weitere Operatoren zum Berechnen und weitere Funktionen. Aber wenn Sie die vorstehenden Erläuterungen nachvollziehen konnten, haben Sie die Programmierung mit LibreOffice Basic grundsätzlich verstanden.

Wie oben schon gesagt, finden Sie für die hier und im Folgenden verwendeten Basic-Konstrukte Dokumentationen und Anleitungen an den Stellen, die in „Weitere Schritte“ auf Seite 36 beschrieben sind.

Makros verwalten

Nachdem der Inhalt des Makros beschrieben wurde, geht es nun darum, das Makro auch auszuführen. Dazu muss das Makro zunächst einmal in LibreOffice eingegeben und gespeichert werden. Wenn Sie eines der Beispielmakros ausprobieren wollen, können Sie es einfach in die Zwischenablage kopieren und in der Basic-IDE, die nun beschrieben wird, einfügen.

Ein Makro hinzufügen

Hier stehen Sie als erstes vor der wichtigen Frage,wo das Makro gespeichert werden soll. LibreOffice kennt drei Typen von Speicherorten von Makros (vgl. Abbildung 9):

- „Meine Makros“: hier speichern Sie Makros, die Sie dokumentunabhängig benötigen. Auch Extensions (siehe das Kapitel „Anpassen von LibreOffice“ in diesem Handbuch) speichern ihre Makros dort; in der Abbildung 9 sind das – auch wenn man das an nichts erkennen kann, denn es ist völlig unerheblich, ob eine Makro durch direkte Eingabe oder durch Installieren einer Extension erzeugt wurde – unter anderem AltSearch, Embedder, VisibleBookmarks und XrayTool.
- „LibreOffice Makros“: hier sind Makros gespeichert, die von LibreOffice bereits bei der Installation geliefert wurden; auch diese Makros sind unabhängig davon, welches Dokument geöffnet ist, verfügbar.
  Dieser Speicherort enthält Bibliotheken, die von LibreOffice genutzt werden (wie z.B. „Access2Base“ oder „FormWizard“), aber auch hilfreiche kleine Module, z.B. in der Bibliothek „Tools“, die Sie in eigenen Programmen aufrufen können, um Ihren eigenen Code kürzer zu halten.
- Im Dokument: erkennbar am Dokumentsymbol vor dem Namen, sehen Sie hier die Dokumente, die gerade geladen sind. Makros, die dort gespeichert sind, sind auch nur dann verfügbar, wenn das Dokument geladen ist. Dieser Speicherort ist also normalerweise nur dann zu empfehlen, wenn es sich um Makros handelt, die speziell mit den Inhalten des Dokuments arbeiten.

Diese Speicherorte werden nun weiter unterteilt in Bibliotheken. Diese dienen dazu, die Makros in zusammenhängende Themenbereich zu gliedern. Im Beispiel sind das zum Teil von außen, durch eine installierte Extension, verursachte Gliederungen, wie „AltSearch“ oder „DirectColourManager“, oder selbst definierte wie „Hilfsmittel“.

Eine Bibliothek wird weiter in Module unterteilt, sichtbar bei „Embedder“ und „Hilfsmittel“, bei denen die Struktur aufgeklappt wurde.

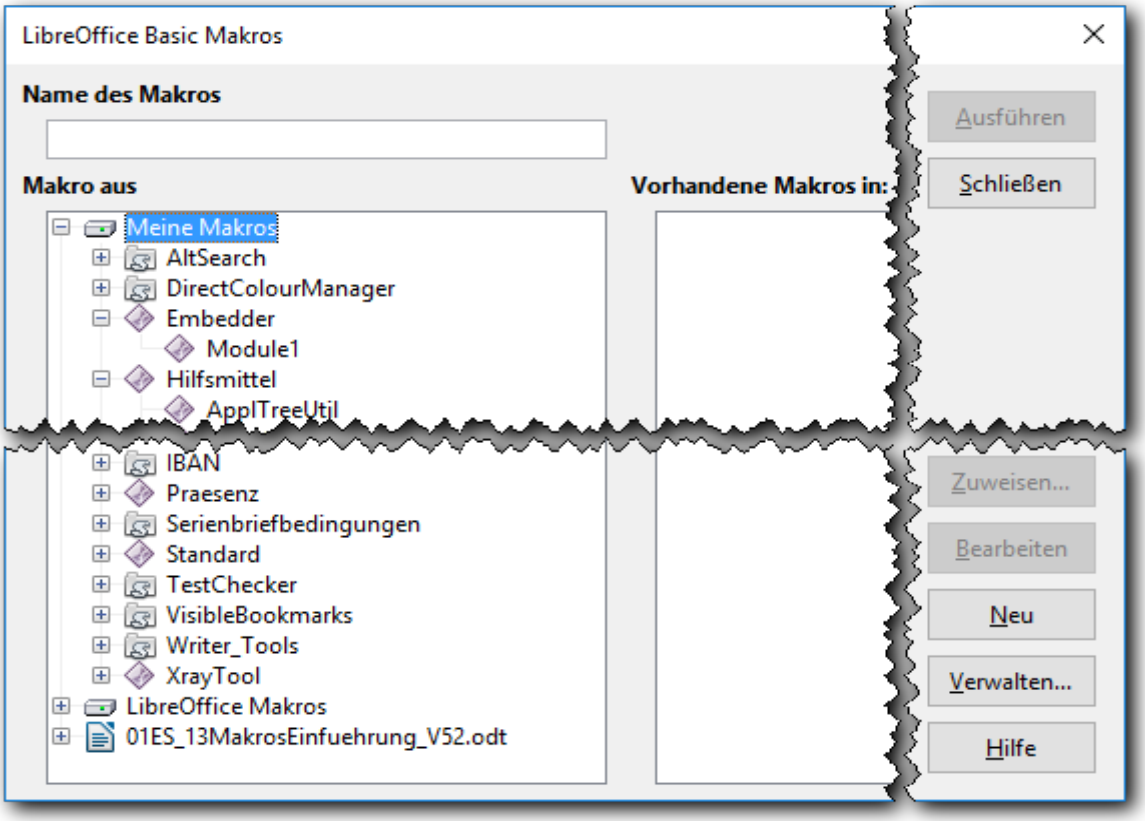

Abbildung 5: Verwalten von Makros

Ein Modul enthält dann ein oder mehrere Makros, was Sie auf der rechten Seite sehen würden, wenn ein Modul markiert wäre.

Die Form des Symbols vor Bibliotheken und Modulen besagt übrigens, ob das Objekt bereits geöffnet ist (Raute) oder nicht.

In unserem Fall liegt es nun nahe, das Makro nicht in einem Dokument, sondern allgemein verfügbar zu speichern, also unter „Meine Makros“. Wir können nun entscheiden, ob wir eine neue Bibliothek anlegen oder das Programm – wahrscheinlich in einem neuen Modul – in einer bestehenden Bibliothek anlegen wollen. Im gezeigten Beispiel würde sich vom Namen her die Bibliothek „Hilfsmittel“ vielleicht anbieten, aber wenn man mit der Programmierung in LibreOffice beginnt, gibt es solche Bibliotheken noch nicht, deshalb wird nun der Weg zur Erstellung einer neuen Bibliothek beschrieben, beim Speichern in einer bestehenden Bibliothek entfällt einfach ein Schritt.

Aufruf der Makroverwaltung

Grundsätzlich öffnen Sie den Dialog für die Verwaltung von Makros mit **Extras → Makros → Makros verwalten → LibreOffice Basic...**

Welche Schaltfläche Sie danach betätigen, hängt von Ihren Absichten ab. Mit **Neu** würden Sie, wenn ein Dokument markiert ist, einen neuen Modul in diesem Dokument anlegen können. Ist al-

lerdings „Meine Makros“ markiert, dann wird ein neues Makro in einem bestehenden Modul angelegt; welcher das ist, würde hier zu sehr ins Detail gehen, da unser Ziel ja eine neue Bibliothek ist.

Erstellen einer Bibliothek

Wählen Sie die Schaltfläche **Verwalten**, dann den Reiter *Bibliotheken* (Abbildung 6)

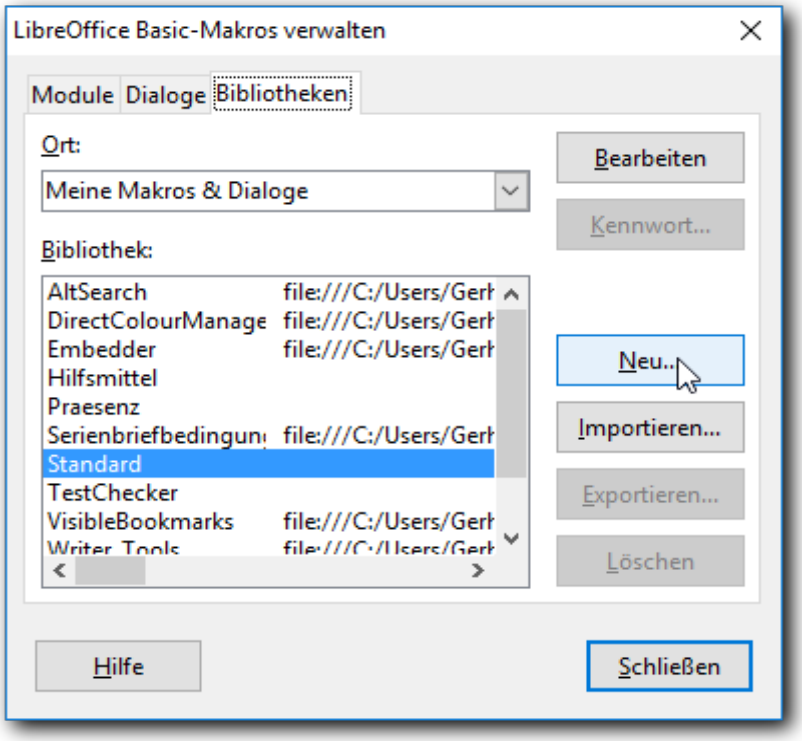

Abbildung 6: Neue Bibliothek

Es ist hier unerheblich, dass standardmäßig die Bibliothek „Standard“ markiert ist, da durch unsere nun folgende Wahl eine neue Bibliothek in „Meine Makros und Dialoge“ angelegt wird. Beachten Sie, dass der nun angezeigte Name, wie wir weiter unten sehen werden, korrekter ist als der vorher sichtbare „Meine Makros“.

Betätigen Sie die Schaltfläche **Neu**. Sie werden aufgefordert, einen Namen für die neue Bibliothek zu vergeben (Abbildung 7):

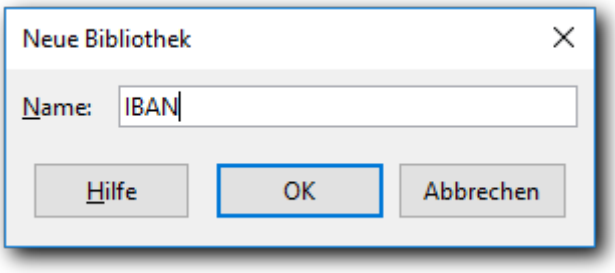

Abbildung 7: Neue Bibliothek: Namen vergeben

Als Ergebnis sehen Sie die neue Bibliothek am Ende der Liste der Bibliotheken (Abbildung 8). Bei einer späteren Anzeige der Liste der Bibliotheken ist sie dann nach dem Alphabet einsortiert.

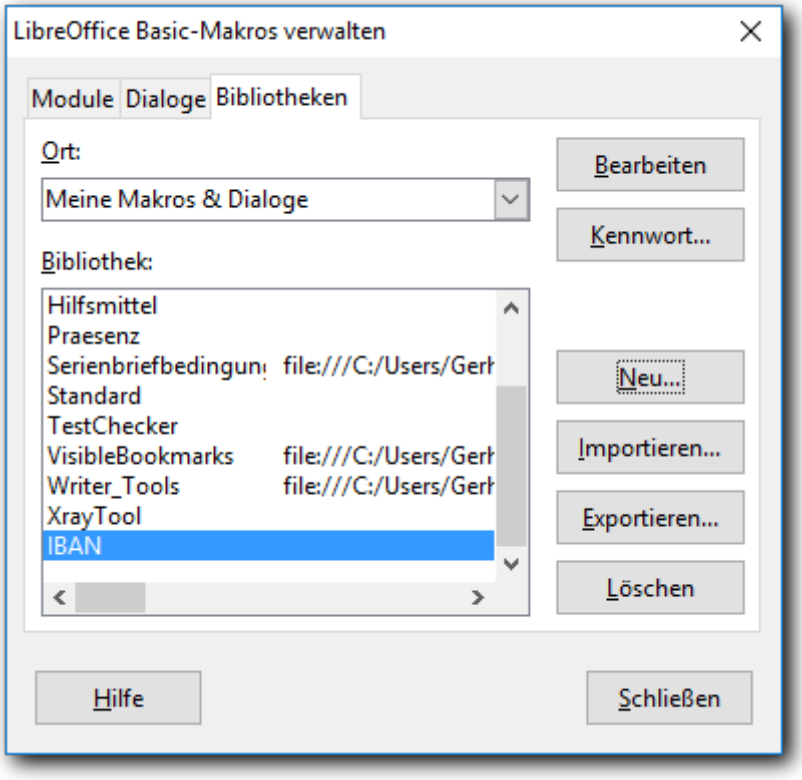

Abbildung 8: Neue Bibliothek wurde angelegt

Erstellen eines Moduls

Wählen Sie nun den Reiter *Module*. Die neue Bibliothek ist bereits markiert (Abbildung 9). Diese Situation würden Sie auch erreichen, wenn Sie eine bestehende Bibliothek auswählen, um dort einen neuen Modul anzulegen. Betätigen Sie die Schaltfläche **Neu**.

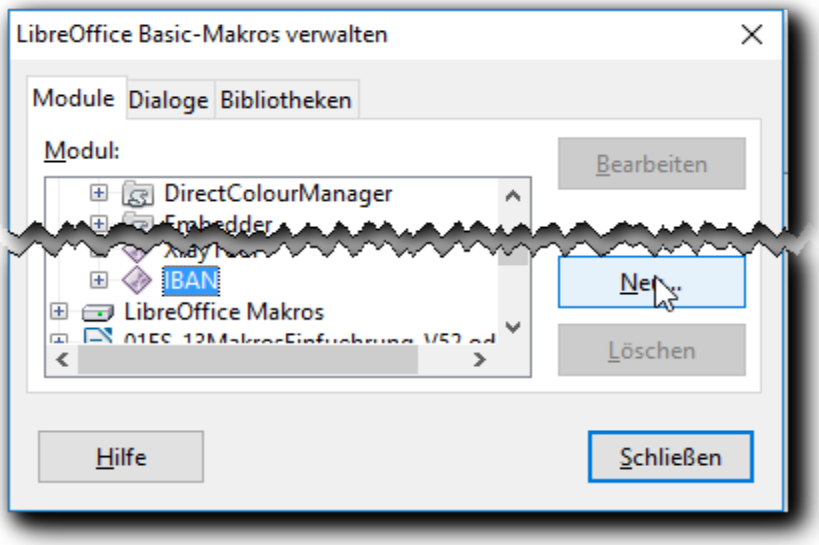

Abbildung 9: Neuen Modul anlegen

Sie werden aufgefordert, einen Namen für den neuen Modul zu vergeben (Abbildung 10).

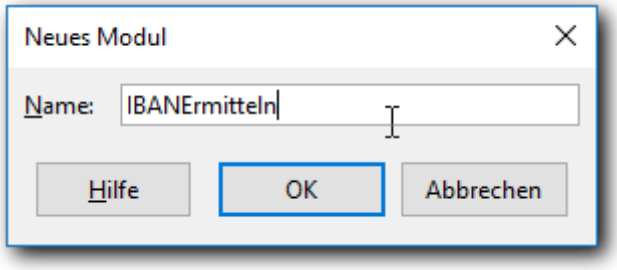

Abbildung 10: Neuen Modul anlegen: Namen vergeben

Als Ergebnis sehen Sie den neuen Modul als Unterpunkt der Bibliothek „IBAN“ (Abbildung 11):

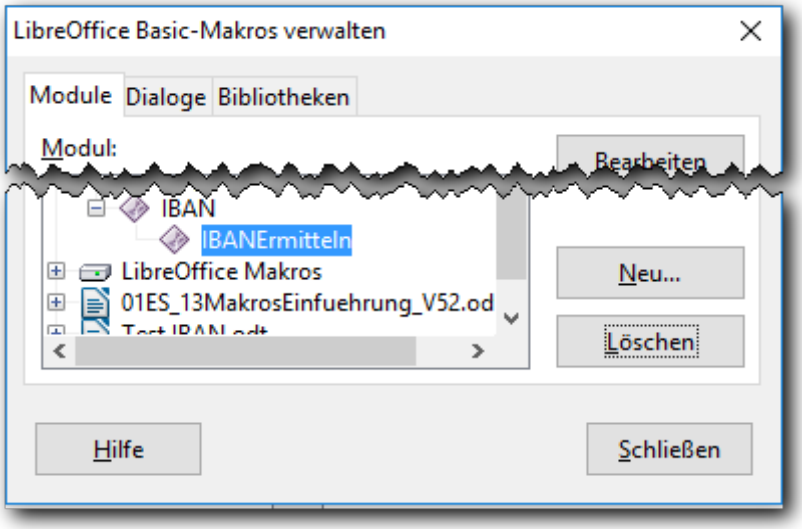

Abbildung 11: Der neu erzeugte Modul

Schließen Sie den Dialog mit **Schließen**.

Im weiterhin geöffneten darunterliegenden Dialog, aus dem wir in Abbildung 5 die folgenden Dialoge aufgerufen hatten, doppelklicken Sie ggf. die Bibliothek „IBAN“ oder klicken Sie auf das Plus-Zeichen vor dem Namen, um den Unterpunkt anzuzeigen. Dann doppelklicken Sie „IBANErmitteln“, um den Inhalt dieses Moduls anzuzeigen.Das Ergebnis zeigt Abbildung 12.

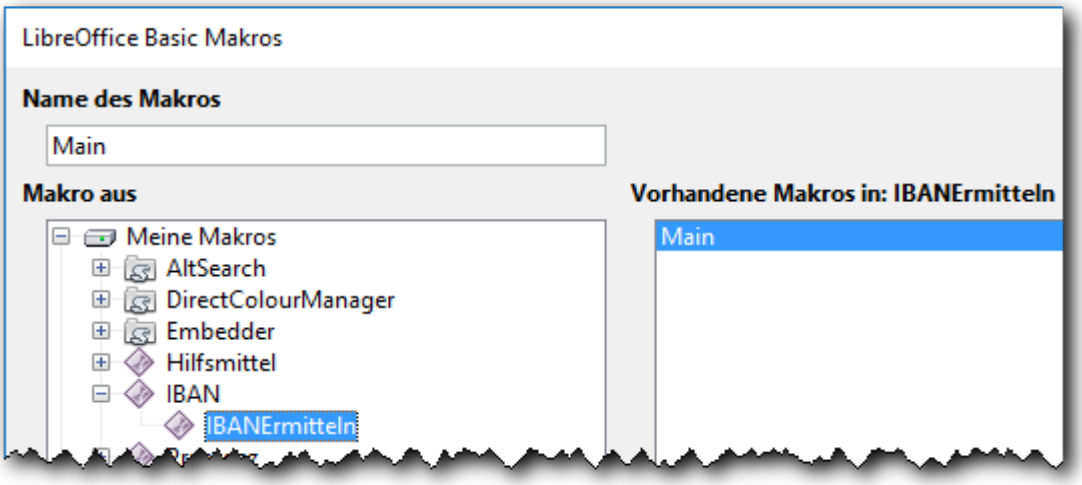

Abbildung 12: neuer Modul: Inhalt ändern

Die Liste rechts zeigt die vorhandenen Makros in diesem Modul. Ein neuer Modul wie unserer hat natürlich noch keine Makros. Daher wird beim Anlegen des Moduls eine sub mit dem Namen „Main“ mitsamt „end sub“, aber ohne dazwischenliegenden Programmcode, angelegt. Diesen Namen sehen Sie in der Liste rechts.Betätigen Sie die Schaltfläche *Bearbeiten*, um den Modul – hier also noch „Main“ – in der Basic-IDE zu öffnen (Abbildung 12).

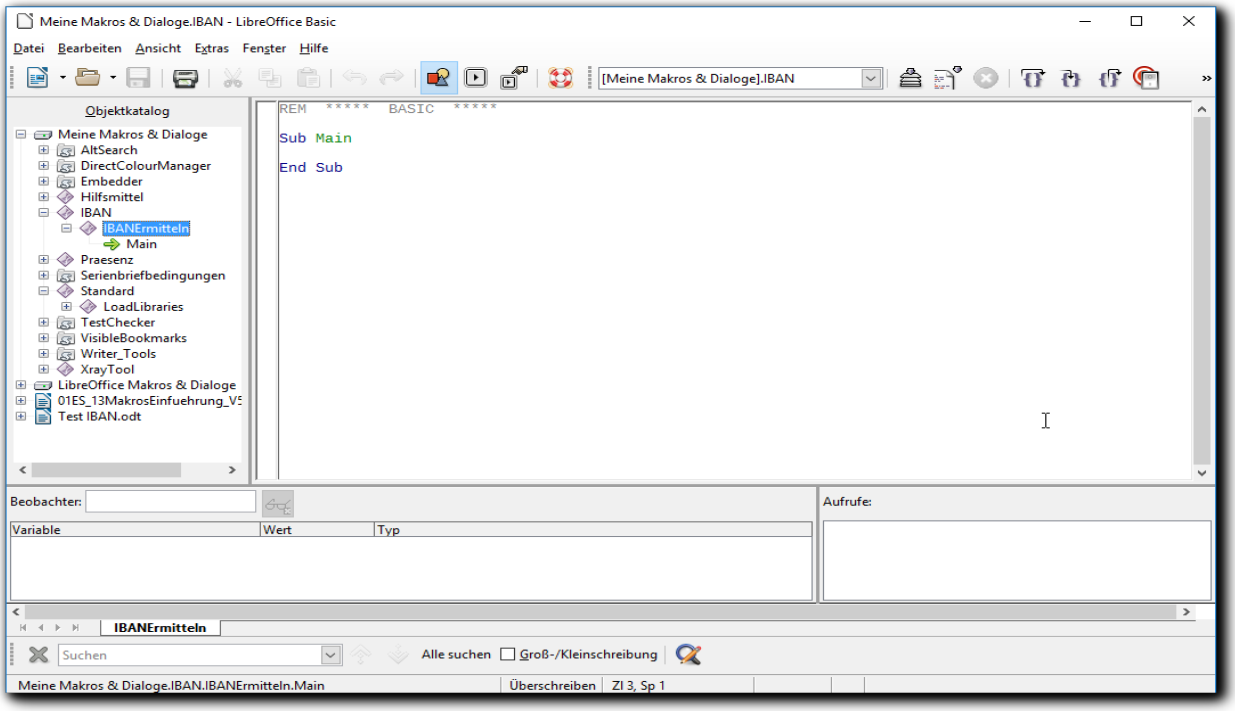

Abbildung 13: die Basic-IDE

Die Basic-IDE

Die IDE (Integrated Development Environment = Integrierte Entwicklungsumgebung) ist das Werkzeug, mit dem Sie Makros schreiben und ändern.

- Im oberen Teil sehen Sie die Menüleiste, auf die hier – bis auf Ausnahmen – nicht im Einzelnen eingegangen wird.
- Darunter sehen Sie im linken Teil eine Baumdarstellung der Speicherorte, Bibliotheken, Module und Programme ähnlich wie wir sie schon kennengelernt haben. Der auffallende Unterschied ist, dass in diesem Kontext auch die einzelnen Makros gezeigt werden, weil es nun um deren Verwaltung geht. Sie sehen im Beispiel unter der markierten Zeile des Moduls „IBANErmitteln“ das genannte Makro „Main“
- Im rechten Teil wird – je nach Aktion - das erste oder das markierte Makro innerhalb des Moduls angezeigt; die Programme sind innerhalb eines Moduls einfach hintereinander, in der Reihenfolge, wie Sie sie anordnen, gespeichert.
- Darunter sehen Sie Fenster, die für die Entwicklung hilfreich sind:
  - den Beobachter, mit dem Sie den Inhalt von Variablen bei der Ausführung des Programms verfolgen können
  - ein Listenfeld zur Verfolgung der Aufrufe
- Unter dem Rollbalken zeigen Reiter die Module an, die die Bibliothek enthält; mit einem Rechtsklick im freien Bereich neben den Modulnamen öffnen Sie ein Menü, mit dem Sie

neue Module und Dialoge anlegen können.

- Als letztes wird noch eine Suche innerhalb des Moduls, mit den auch aus den anderen Komponenten von LibreOffice bekannten Optionen, angeboten.

Tipp	Die Anzeige der Suche und der Baumdarstellung der Makrostruktur und weiteres können Sie In der Basic-IDE über den Menüpunkt **Ansicht** beeinflussen.

Sie können nun, wenn Sie wollen, den Text des Makros aus Beispiel 2 (oder auch Beispiel 1, wenn Sie selber den Fehler sehen wollen) in die Zwischenablage kopieren und im Modul *IBANErmitteln* einfügen. Das Makro Main bleibt dabei, wenn Sie es nicht überschrieben haben, erhalten, was nicht schadet, da es keinen Befehl enthält, der etwas auslöst. Da wir aber aussagekräftige Namen für unsere Makros vorziehen, werden wir mit diesem Rumpfmakro sowieso nicht arbeiten und es am besten gleich löschen. Dass es überhaupt erzeugt wird, hat sowohl praktische als auch historische Gründe.

Ein Makro ausführen

Wenn Sie nun ein Makro eingegeben oder eingefügt haben, können Sie es aus der IDE auch gleich direkt ausführen und testen. Betätigen Sie dazu das Symbol  .

Hinweis	Wenn der Modul mehrere Makros enthält, wird das Makro ausgeführt, in dem sich der Cursor befindet.

Wenn Sie sich nicht in der IDE befinden, gibt es mehrere Möglichkeiten, ein Makro auszuführen:

- Über die Menüauswahl **Extras → Makros → Makro ausführen…**
  Es wird dann der Dialog *Makro-Auswahl* geöffnet (Abbildung 14), in dem Sie die gewünschte Bibliothek und den gewünschten Modul durch Aufklappen der Baumstruktur wählen können. Im rechten Listenfeld können Sie dann in der Liste der in diesem Modul enthaltenen Makros das gewünschte auswählen. Mit der Schaltfläche **Ausführen** starten Sie das gewählte Makro.

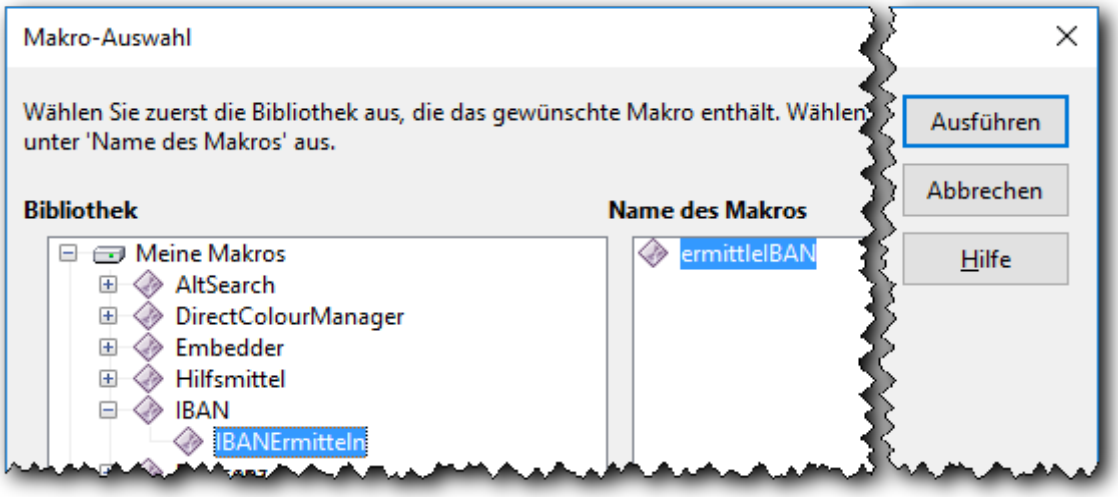

Abbildung 14: Ausführen eines Makros über Makro-Auswahl

- auch über die Verwaltung von Makros können Sie ein Makro starten: der Dialog zur Verwaltung von Makros (Abbildung 5) ist ganz ähnlich zu dem gerade gezeigten aufgebaut

und enthält ebenfalls die Schaltfläche **Ausführen**, die aktiv wird, wenn Sie wie gerade beschrieben ein Makro ausgewählt haben

- Sie können – was vor allem für sehr häufig benötigte Makros sinnvoll sein kann – die Ausführung des Makros einer Tastenkombination, einem Ereignis des Dokuments (das sind wichtige Aktionsschritte bei der Arbeit mit einem Dokument, wie Öffnen oder Speichern), oder einem (neuen) Symbolleisten- oder Menüeintrag zuordnen. Mehr dazu im Kapitel „LibreOffice anpassen“ dieses Handbuchs.
- Sie können in Ihrem Dokument oder in Ihrer Anwendung – d.h. in deren Dokumenten, Formularen oder Dialogen – die Ausführung des Makros einer selbst definierten Schaltfläche zuordnen.

Erstellen von Makros, Fortsetzung

Wir wollen nun das Beispiel weiter ausbauen, um aus der Sphäre eines Testbeispiels langsam in die einer echten Anwendung überzugehen.

Beispiel 3: IBAN ermitteln mit Prüfungen, Unterprogrammaufruf

Das bisher erstellte Makro hat noch einige Schwachstellen,die sich im dauerhaften Betrieb herausstellen:

- Das Makro funktioniert nur, wenn man Bankleitzahl und Kontonummer im Code selbst eingibt; es sollte aber immer wieder mit unterschiedlichen Werten für diese beiden Parameter eingesetzt werden können.
- Tippfehler bei Bankleitzahl oder Kontonummer lassen sich zwar nie ganz vermeiden; aber wenn Buchstaben hineingeraten, geht die Umwandlung in eine Zahl entweder schief oder ist unvollständig
- Kontonummern sind oft kürzer als 10 Stellen, bei der Eingabe ist man gewohnt, keine führenden Nullen anzugeben.

Diese Punkte werden in der neuen Version berücksichtigt:

- Das Programm wurde in zwei Komponenten aufgeteilt:
  - das Hauptprogramm (das eigentlich nur zum Testen dient, denn später werden entsprechende Programme, die noch andere Logik beinhalten, es ersetzen) setzt die Parameter zum Aufruf des Unterprogramms und wertet dessen Ergebnis aus; es ist somit der erste, vorläufige Prototyp eines Programms, das das wiederverwendbare, universelle Unterprogramm aufruft.
  - das Unterprogramm, das vom Hauptprogramm gerufen wird, beinhaltet die Logik, die wir schon entwickelt haben, plus die oben genannten zusätzlichen Prüfungen.

Die – nun beiden – Programme haben nun folgendes Aussehen; die Einzelheiten werden im Anschluss beschrieben:

```
'  Hauptprogramm zum Testen des Unterprogramms
sub ermittleIBAN1

dim erfolg as boolean, ergebnis as string

erfolg = FuncErmittleIBAN("79876543", "12345678", ergebnis)
if erfolg then
   msgbox (ergebnis, 64, "ermittelte IBAN")
 else
```

```
   'hier koennte eine Reaktion im Fehlerfall erfolgen; die Meldung ist
   'allerdings schon im Unterprogramm erfolgt.
end if

end sub

'  Universell verwendbares Unterprogramm
function funcErmittleIBAN (blz as string, konto as string, _
      iban as string) as boolean
REM vereinfachte Variante: nur für Deutschland, nur Standardfälle
REM  abgedeckt;   Quelle: deutsche wikipedia

dim ibanBasis as string, rest as long, i as long

funcErmittleIBAN = False
if Not isNumeric (blz) then
   msgbox ("Bankleitzahl " & blz & " ist nicht numerisch", 16, _
      "Numerisch-Prüfung")
   exit function
end if
if Not isNumeric (konto) then
   msgbox ("Kontonummer " & konto & " ist nicht numerisch", 16, _
      "Numerisch-Prüfung")
   exit function
 else
   konto = Format(konto , "0000000000")
end if
funcErmittleIBAN = True

ibanBasis = blz & konto & "131400"  'die letzten Stellen ...
for i = 1 to 24
   rest = (rest * 10 + Mid(ibanBasis,i,1)) MOD 97
next
iban = "DE" & Format(98 - rest, "00") & blz & konto

end function
```

Die Programmzeilen werden nun in der Reihenfolge besprochen, wie sie auch beim Aufruf durchlaufen werden. Der Aufruf muss für *ermittleIBAN1* erfolgen, das ist ja hier das Hauptprogramm, wo wir die gewünschten Parameter setzen; wenn Sie versuchen, *FuncErmittleIBAN* zu starten, werden Sie eine Fehlermeldung erhalten, weil gewisse Variablen, nämlich die Parameter, nicht gesetzt sind.

- Als Prototyp für noch zu erstellende Programme, die sich mit der Ermittlung der IBAN beschäftigen sollen, wurde ein kurzes Programm erstellt, in dem nun Bankleitzahl und Kontonummer gesetzt werden, die das gerufene Unterprogramm empfängt. Damit ist das Unterprogramm unabhängig von konkreten Werten und kann immer wieder mit andern Parametern aufgerufen werden.
  Das Programm heißt einfach *ermittleIBAN1* (es ist ja nur ein Testprogramm).
- es ruft das Unterprogramm *FuncErmittleIBAN* in der Zeile „erfolg = ...“; dazu unten mehr. Damit diese neue Programm, das zwar aus *ermittleIBAN* entstanden ist, aber doch ein bisschen anders aussieht, nicht mit diesem verwechselt wird, das ja hier im gleichen Dokument steht, haben wir ihm einen anderen, wenn auch etwas sperrigen Namen gegeben. In der Praxis wird man das zweite aus dem ersten entwickeln und das erste daher nicht mehr benötigen und kann deshalb den Namen beibehalten. Da dieses neue Programm nun nicht mehr die Bankleitzahl und Kontonummer direkt per Zuweisung erhält, müssen diese Werte

vom aufrufenden Programm zur Verfügung gestellt werden. Dies geschieht, indem man sie als Parameter in der Definition des Unterprogramms *FuncErmittleIBAN* definiert: die beiden Variablen stehen nun nicht mehr in einer dim-Anweisung – was bedeutet, dass sie lokal innerhalb dieses Programms definiert sind – sondern in Klammern hinter dem Programmnamen, was heißt, dass sie von dem aufrufenden Programm – im Moment ermittleIBAN1, zukünftige Programme müssen dann den Aufruf entsprechend implementieren – geliefert werden.

- Beim gerufenen Programm *FuncErmittleIBAN* fällt noch etwas auf: die Zeile nach den beiden genannten Parametern endet nach einem Leerzeichen mit einem Unterstrich. Dieser Unterstrich ist das Zeichen, dass der Befehl auf der nächsten Zeile weitergeht: die Fortsetzung des Befehls wurde aus Platzgründen in diesem Dokument umgebrochen. Im Beispiel wird der dritte Parameter und auch der Rückgabewert der Funktion auf einer neuen Zeile benannt.
  Zeilen in der IDE können sehr lang sein, ein Umbruch ist aus technischen Gründen praktisch nicht nötig, wohl aber wegen der Lesbarkeit, um das Rollen in der IDE-Ansicht zu vermeiden.

- Beim Programm *FuncErmittleIBAN* wurde außerdem das Schlüsselwort „sub“ durch „function“ ersetzt. Das ist die zweite Variante für die erste Zeile eine Makros, die oben schon erwähnt wurde: eine function führt nicht nur – genau wie die sub – den Code aus, sondern liefert auch direkt einen Wert zurück, der normalerweise als erstes ausgewertet wird und üblicherweise die grundsätzliche Aussage über das Ergebnis des Unterprogramms beinhaltet.
  Die Gestaltung der Schnittstelle (d.h. welche Werte wie übergeben werden) liegt natürlich im Ermessen des Programmierers, aber häufig üblich ist, dass der Erfolg des gerufenen Programms als Boolesche Variable, also mit den Werten „ja“ oder „nein“) oder in komplizierteren Fällen auch als numerische oder alphanumerische Variable geliefert wird, und dass der eventuelle Ergebniswert als weiterer Parameter definiert wird. In diesem Sinn wurde in unserem Beispiel das Ergebnis des Unterprogramms *FuncErmittleIBAN* – die IBAN – als dritter Parameter definiert. Hinter den Klammern, die die Parameter der Funktion *FuncErmittleIBAN* definieren, wurde mit „as boolean“ festgelegt, dass die Funktion als Rückgabewert eine boolesche Variable liefern soll.

- Im Code von *FuncErmittleIBAN* muss nun auch dafür gesorgt werden, dass dieser boolesche Wert gesetzt wird. Dies geschieht, indem man dem Namen der Funktion den Wert „True“ = „wahr“ oder „False“ = „falsch“ zuweist. Hat der Rückgabewert einen anderen Typ, sind natürlich andere Rückgabewerte zu setzen. Im Beispiel – aber da beginnt schon der persönliche Programmierstil – wird vor Beginn der Prüfungen erst einmal der Erfolg auf „fehlerhaft“ gesetzt, bei Bestehen der Prüfungen wird dieser Zustand auf „erfolgreich“ geändert.

- In den Deklarationen („dim ...“) fehlen nun natürlich alle Variablen, die bereits als Parameter definiert wurden.

- Für die Felder Bankleitzahl und Kontonummer, die beide numerisch sein sollten, erfolgt nun eine Prüfung mit der Funktion isNumeric, da nichtnumerische Werte Probleme bei der Ermittlung der IBAN nach sich ziehen; bei nichtnumerischen Werten erfolgt eine Meldung und die Beendung des Programms durch den Befehl „exit function“.

- Die Prüfung erfolgt mit der üblichsten Form einer sogenannten Verzweigung, bei der abhängig vom Inhalt einer Variablen verschiedene (Folgen von) Aktionen durchgeführt werden.

  - Hinter „if“ wird eine Bedingung genannt, die ausgewertet wird, was dann als Ergebnis einen booleschen Wert liefert; die Funktion *isNumeric* liefert „True“ („wahr“), wenn die Variable numerisch ist, die Verneinung „Not isNumeric“ liefert also „wahr“, wenn die Variable nicht numerisch ist.

- Hinter „then“ wird nun die (Folge von) Befehlen aufgeführt, die ausgeführt werden soll, wenn die Bedingung wahr ist (im Beispiel, wenn die Variable nicht numerisch ist); hier wird die bereits bekannte Messagebox verwendet, der zweite Parameter „16“ bewirkt ein anderes Symbol.
- Hinter dem Schlüsselwort „else“ werden die Befehle benannt, die ausgeführt werden sollen, wenn die Bedingung nicht zutrifft. Im Beispiel wird bei der zweiten Prüfung bei der Kontonummer eine Umwandlung mittels der Funktion „Format“ durchgeführt, die wir weiter oben schon kennengelernt haben; sie sorgt dafür, dass die Kontonummer mit der benötigten Anzahl von führenden Nullen aufgefüllt wird.

- Nach den Prüfungen wird der Erfolg des Programms auf „True“ gesetzt und die IBAN ermittelt; das Ergebnis wird nicht wie bisher auf den Bildschirm ausgegeben, sondern dem rufenden Programm per drittem Parameter übergeben.
- Die Schleife ist nun so vereinfacht wie im Tipp in „Beispiel 2: Das korrigierte Makro“ auf Seite 11ff. beschrieben.
- Im Programm *ermittleIBAN1* wird in der Zeile „erfolg = ...“ die Funktion *FuncErmittleIBAN* aufgerufen. Die Parameter wurden schon oben besprochen, desgleichen, dass die Funktion einen Rückgabewert liefert, der als boolesche Variable die Werte „True“ oder „False“ annehmen kann. Dieser Rückgabewert wird nun der Variablen erfolg (des Programms *ermittleIBAN1*) zugewiesen, damit er in der folgenden Anweisung abgefragt werden kann. Wieder wird die Verzweigung mit „if … then … else … end if“ verwendet.
  - Da der Rückgabewert von *FuncErmittleIBAN* bereits einen booleschen Wert liefert, genügt es, die Variable selbst zu verwenden statt sie mit … = True abzufragen.
  - Hinter „then“ wird die bereits bekannte Messagebox zur Ausgabe der ermittelten IBAN verwendet. Der Unterschied zum früheren Beispiel besteht darin, dass diese Ausgabe nun nur erfolgt, wenn *FuncErmittleIBAN* keinen Fehler fand.
  - Hinter dem Schlüsselwort „else“ werden in unserem Beispiel keine Befehle benannt, die ausgeführt werden sollen, weil wir damit zufrieden sind, dass das gerufene Programm *FuncErmittleIBAN* bereits die Fehlerursache gemeldet hat. Das Schlüsselwort „else“ kann in einem solchen Fall, wo nichts weiter passieren soll, auch einfach weggelassen werden.

Die Ausgabe der neuen Kombination der beiden Makros sieht im Normalfall nicht anders aus als vorher (siehe Abbildung 4), allerdings führen die Eingabefehler nun nicht mehr zu eventuell rätselhaften Abbrüchen, sondern zu verständlichen Meldungen wie in Abbildung 15.

Wie die Trennung in Haupt- und Unterprogramm gewinnbringend eingesetzt wird, sehen wir nun in den beiden folgenden Beispielen.

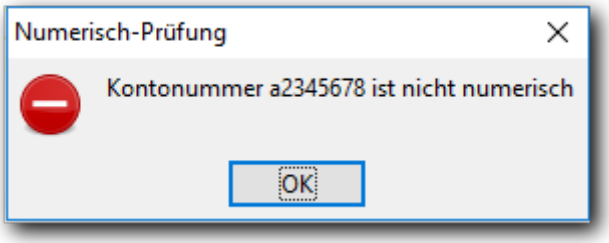

Abbildung 15: Fehlermeldung für nichtnumerische Kontonummer

Beispiel 4: Massensetzung der IBAN in einer Calc-Tabelle

Kontoverbindungen hat man öfters gespeichert, in manchen Fällen viele z.B. in einer Calc-Tabelle, z.B. für ein kleines Unternehmen oder einen Verein. Unser Unterprogramm *FuncErmittleIBAN* kann in einem solchen Fall für alle Zeilen der Tabelle aufgerufen werden und jeweils zur Kontoverbindung die zugehörige IBAN ermitteln und in die vorgesehene Spalte der Tabelle eintragen. Das soll das folgende Beispiel illustrieren.

Hinweis	Da, wie schon gesagt, das Programm nur Normalfälle der IBAN-Ermittlung behandelt, kann die ermittelte IBAN nur ein Vorschlag für die vom Adressaten zu bestätigende tatsächliche IBAN sein, was ihm und Ihnen aber in fast allen Fällen manuelles Abschreiben erspart.

Als Beispiel für eine Calc-Tabelle, die auf diese Weise bearbeitet werden soll, wurde die folgende Testtabelle erstellt, die wirklich nur die notwendigen Spalten und auch nur zwei Zeilen enthält (Abbildung 16):

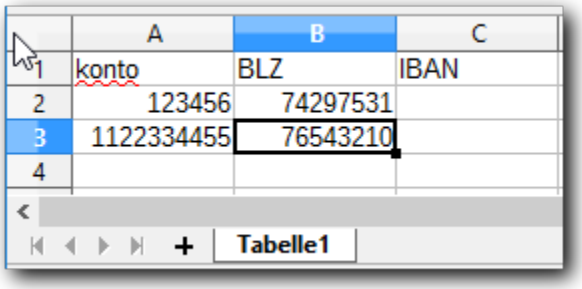

Abbildung 16: Testttabelle für IBAN-Berechnung vor der Berechnung

Aufgabenstellung

Das kurze Programm, das wir nun erstellen werden, soll vom Benutzer die Unter- und die Obergrenze für die Zeilennummern erfragen, für die die IBAN ermittelt und in die dafür vorgesehene Spalte eingetragen werden soll. Die Ausführung für einen Bereich von Zeilen erlaubt es, bei einem erneuten Aufruf, z.B. weil bei einer Zeile ein Problem auftrat, beispielsweise ein Buchstabe in der Kontonummer, nicht immer wieder von vorne beginnen zu müssen.

Diese Aufgabenstellung ermöglicht es, für die zwei großen Themenbereiche der Makroprogrammierung, die noch nicht angeschnitten wurden, einen ersten Einblick zu geben:

- den Zugriff auf die Objekte von UNO (s. „Objekte von LibreOffice“ auf Seite 6)
- den Einsatz von Dialogen

Anlegen des Makros

Das geplante Makro ist, in der einfachen Form, wie wir es hier erstellen, nur für unsere Testtabelle einsetzbar, daher speichern wir es in dem Calc-Dokument.

Der Weg ist im Grunde gleiche wie oben in auf Seite „Makros verwalten“ auf Seite 13 beschrieben, nur dass wir keine neue Bibliothek anlegen, sondern die standardmäßig vorhandene Bibliothek „Standard“ verwenden.

Wählen Sie wieder **Extras → Makros → Makros verwalten → LibreOffice Basic…,** klappen Sie das Calc-Dokument in der Baumstruktur links auf, wählen Sie die Bibliothek „Standard“ aus und

betätigen Sie die Schaltfläche **Neu**. Sie werden wie in Abbildung 10 aufgefordert, einen Namen für den nun zu erstellenden Modul zu vergeben. In unserem Testbeispiel haben wir der Einfachheit halber auf einen selbstgewählten Namen verzichtet und den vorgeschlagenen Namen „Module1“ belassen.

In der IDE wird nun nach der Eingabe des Namens und **OK** gleich der neue Modul geöffnet, er enthält wieder die leere sub mit dem Namen Main.

Hier können Sie nun mit dem Schreiben des Makros beginnen, aber irgendwann benötigen Sie in diesem Programm Angaben bezüglich des Dialogs, der zur Eingabe der Unter- und Obergrenze für die Zeilennummern dient. Wir erstellen daher als erstes den Dialog und wenden uns dann wieder dem Makro zu.

Erstellen des Dialogs

Es würde den Rahmen dieser Einführung sprengen, das Gestalten eines Dialogs detailliert darzustellen, aber die kurzgefassten Andeutungen sollten Ihnen einen Eindruck vermitteln, dass das schnell und einfach zu bewerkstelligen ist.

Der Dialog hat in der Verwaltungssicht folgendes Aussehen (Abbildung 17):

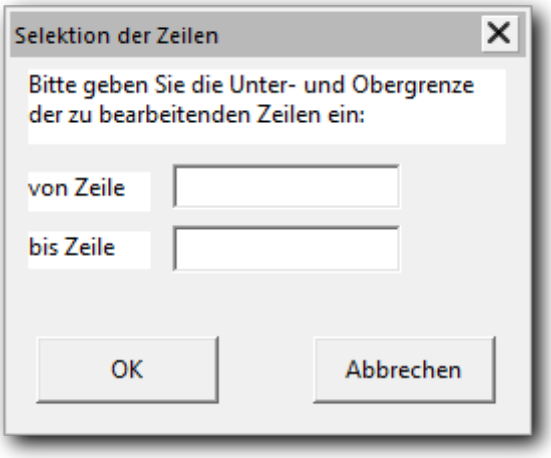

Abbildung 17: Der Dialog für Unter- und Obergrenze

Um einen Dialog anzulegen, klicken Sie in der IDE mit der rechten Maustaste in das freie Feld rechts neben dem oder den Modul- und Dialognamen (im Beispiel ist das nur nur „Module1“) und wählen Sie in dem nun geöffneten Kontextmenü *Einfügen*. Sie können dann wählen, ob Sie einen neuen Modul oder – in unserem Fall – einen neuen Dialog anlegen wollen (Abbildung 18). Der neue Dialog erhält automatisch einen vom System vergebenen Namen, der als weiterer Reiter bei den Modul- und Dialognamen erscheint. Dort erscheinen zunächst alle Module, dann alle Dialoge, jeweils in alphabetischer Reihenfolge.

Für die verschiedenen Kontrollelemente, die Sie in einem Dialog verwenden können, gibt es nun eine Symbolleiste, in der Sie eines auswählen und dann innerhalb des vorgegebenen leeren Dialogfensters mit der Maus ein Rechteck aufziehen können, in dem das Kontrollelement platziert wird. Anschließend können Sie Größe und Position des Kontrollelements mit der Maus oder mit Tasten verändern, außerdem die gewünschten Werte auch, genauso wie weitere wie den Namen und ggf. Titel und vieles andere, durch direkte Eingabe setzen.

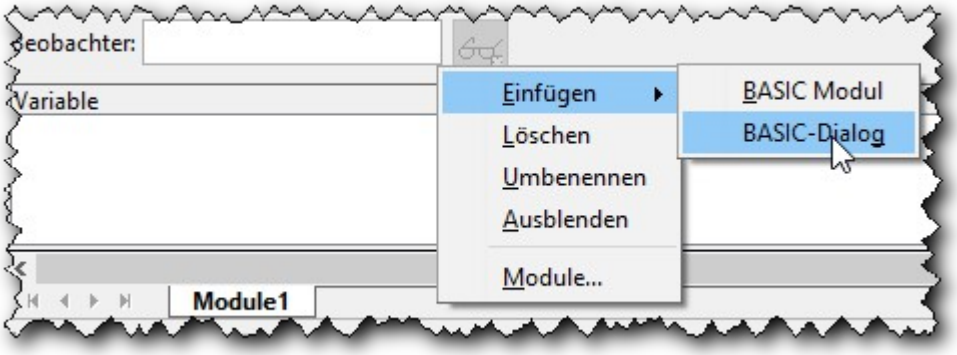

Abbildung 18: Anlegen eines weiteren Moduls oder eines Dialogs

In unserem Fall haben wir folgendes getan:

- zwei numerische Eingabefelder für die Grenzen eingefügt. Durch diese Wahl ersparen wir uns eine eigene Prüfung; wir wollen ja die Zeilennummern zwischen den beiden Werten durchlaufen und ein nichtnumerischer Wert würde zum Programmabbruch führen.
  Bei diesen beiden Feldern wurde noch die Anzahl der Nachkommastellen auf 0 gestellt, weil es sich um Zeilennummern handelt, und ein Name vergeben, weil wir das Feld ja im Programm verwenden werden und ein aussagekräftiger Name dann nützlich ist. Die Namen sind „ZeileVon“ und „ZeileBis“; wir werden Sie im Makro wiederfinden.
- Drei Beschriftungsfelder eingefügt, für die Beschreibung, was im Dialog zu tun ist, und die Benennung der beiden Eingabefelder
- zwei Schaltflächen eingefügt, wie sie in Dialogen üblich sind. Auch für solche Schaltflächen könnten Sie Makros schreiben, die bei deren Betätigung aufgerufen werden, wir haben jedoch auf die vorhandenen Standards zurückgegriffen und der linken auch die Art „OK“ und der rechten die Art „Abbrechen“ zugewiesen. Das bewirkt, dass ihre Betätigung jeweils den Dialog beendet, und zwar:
  - bei „OK“: die Eingaben in alle Felder werden wirksam und können anschließend verwendet werden; das Ergebnis des Dialogaufrufs ist „True“ (wir werden im Makro sehen, wie das verwendet werden kann)
  - bei „Abbrechen“: die Eingaben in die Felder sind unwirksam, das Ergebnis des Dialogs ist „False“
- den Titel des Dialogs gesetzt
- der Name des Dialogs, der durch das Anlegen vom System auf „Dialog 1“ gesetzt worden war, haben wir über das Kontextmenü des Reiters auf „DlgSel“ geändert.

Arbeiten mit UNO-Objekten

Wie in „Objekte von LibreOffice“ auf Seite 6 schon angesprochen, stellt LibreOffice für seine Konstrukte, mit denen man in Makros arbeiten möchte, Zugriffsmöglichkeiten in Form von Objekten auf der Basis der UNO-Architektur bereit. In unserem Makro, wie in vielen anderen Fälle auch, ist das erste verwendete Objekt *ThisComponent*. Dieses Objekt ermöglicht den Zugriff auf das aktive Dokument – in unserem Fall eine Calc-Dokument – mit all seinen Eigenschaften, und auf Methoden, mit denen man etwas bezüglich des Dokuments auslösen kann.

In beiden Fällen können das einfache oder komplizierte Sachverhalte sein:

- bei den Eigenschaften gibt es string-Variablen wie den Titel des Dokuments oder den Filenamen, also den Speicherort, aber auch komplexe „Eigenschaften“ wie *Sheets*, die Kollektion der Arbeitsblätter in diesem Dokument, oder *CurrentSelection*, das die vom

Benutzer vorgenommene Auswahl im Arbeitsblatt beschreibt, die ja aus mehreren Rechtecken von Zellen oder auch aus einer einzelnen Zelle, in der der Cursor steht bestehen kann. Eine weitere „Eigenschaft“, die wir in unserem Programm verwenden werden, ist der *CurrentController*. Das ist ein zum Dokument gehöriges Objekt, das die Informationen bereithält, die zur derzeitigen Anzeige und nicht zur dauerhaften Eigenheit des Dokuments gehören, also z.B. die Selektion, das gerade angezeigte Arbeitsblatt, die Maße der Bildschirmpräsentation usw. Sie sehen im Beispiel, dass mit diesen „Eigenschaften“ auch die mit dem jeweiligen Objekt verknüpfbaren Objekte angeboten werden.

- Bei den Methoden gibt es vom Aussehen her einfache wie *calculate*, das die Berechnung aller Formeln, die im Calc-Dokument vorhanden sind, anstößt, etwas aufwändiger aussehende wie *protect*, denen das Kennwort mitgegeben werden muss, und kompliziert erscheinende wie *print*, die eine ganze Reihe von Parametern erwartet, wobei die Anzahl der notwendigen je nach Druckweg unterschiedlich ist.

Um ein solches Objekt verwenden zu können, weist man es einfach einer Variablen zu, die man in einer dim-Anweisung mit „as object“ deklariert hat:

```
dim calcDoc as object
calcDoc = ThisComponent
```

Eine Eigenschaft oder Methode des Objekts, die man verwenden will, spricht man an, indem man sie, getrennt durch einen Punkt, hinter dem Namen der Variablen anfügt:

```
docTitel = calcDoc.Title   'Zuweisung zu der Variablen docTitel
calcDoc.calculate     'das Berechnen aller Formeln wird ausgelöst
```

Wenn man nur ein einzelnes Attribut oder eine einzige Methode eines Objekts benötigt, kann man sich den Zwischenschritt der Variablen für dieses Objekt sparen:

```
docTitel = ThisComponent.Title
```

Da eine „Eigenschaft“ teilweise wieder ein Objekt ist und wiederum Eigenschaften und Methoden hat, kann man solche Zugriffe auch schachteln:

```
calcDoc.CurrentController.ActiveSheet
```

was über den CurrentController, der die Bildschirmansicht des Dokuments kennt, das derzeit verwendete Arbeitsblatt identifiziert; das verwenden wir in unserem Makro.

Das Makro

Nun stellen wir zunächst das neue Makro vor und erläutern die einzelnen Schritte im Anschluss.

Bei der Zuweisung dlgSel = … haben wir wieder die Zeilentrennung mit „_“ verwendet, aber nur, um auch bei der hier verwendeten Schrift eine korrekte Syntax zu präsentieren; in der IDE passt das gut auf eine Zeile auch bei schmalen Bildschirmen.

```
Sub setIBAN

dim calcDoc as object, sheet as object, cKonto as object
dim cBLZ as object, cIBAN as object
dim dlgSel as object, von as long, bis as long
dim i as long, IBAN as string

calcDoc = ThisComponent
sheet = calcDoc.currentController.ActiveSheet
DialogLibraries.LoadLibrary("Standard")
dlgSel = createUnoDialog(DialogLibraries.GetByName("Standard")._
     getByName("DlgSel"))
if dlgSel.execute then
   von = dlgSel.getControl("ZeileVon").text
   bis = dlgSel.getControl("ZeileBis").text
```

```
   for i = von to bis
      cKonto = sheet.getCellRangeByName("A" & i)
      cBLZ   = sheet.getCellRangeByName("B" & i)
      cIBAN  = sheet.getCellRangeByName("C" & i)
      if cBLZ.string > "" AND cKonto.string > "" then
         if FuncErmittleIBAN(cBLZ.string, cKonto.string, IBAN) then
            cIBAN.string = IBAN
           else
            exit sub'Fehlermeldung erfolgte im gerufenen Programm
         end if
      end if
   next
  else  'Schaltfläche "Abbrechen" oder "Schließen" betätigt
   exit sub
end if

End Sub
```

- Nach der Deklaration der notwendigen Variablen werden, wie im vorigen Abschnitt schon beschrieben, die Variablen für das Dokument und das aktive Arbeitsblatt zugewiesen.
- Nun wird die Bibliothek geladen, die den Dialog enthält. Dadurch, dass wir gerade erst den Dialog erstellt haben, ist diese Bibliothek zwar schon geladen, aber zu einem späteren Zeitpunkt, wo wir das Programm wieder benötigen, muss das nicht der Fall sein. Daraufhin wird mit der Variablen dlgSel eine Instanz des unter dem Namen „DlgSel“ – beachten Sie die unterschiedliche Schreibweise, es sind zwei unabhängige Benennungen – kreiert. Hier wird im einzigen Parameter der Funktion createUnoDialog wieder die Schachtelung von Methodenaufrufen mit jeweils einem Parameter praktiziert.
- Mit der Methode `execute` wird der Dialog ausgeführt, d.h. er wird aufgerufen und wartet darauf, dass der Benutzer – in unserem Fall – die OK- oder Abbrechen-Schaltfläche betätigt. Solange er dies nicht getan hat, geht es im Makro nicht weiter.
- Wenn der Benutzer eine der beiden Schaltflächen betätigt, liefert das Makro, wie oben beschrieben, als Rückgabewert „True“ oder „False“ zurück. Diese Rückgabe fragen wir ab, um zu erfahren, was der Benutzer wollte, und da es sich bereits um einen booleschen Wert handelt, können wir ihn direkt in der if-Anweisung verwenden: wenn „true“ geliefert wurde, also die OK-Schaltfläche betätigt wurde, dann werden die Aktionen bis hin zum Eintragen der IBAN ausgeführt, andernfalls (also bei Abbruch) wird das Makro mit `exit sub` verlassen: nichts wird am Dokument verändert.
- Im Fall, dass der Dialog mit „True“ endete, wird als erstes die Eingabe in den beiden Dialogfeldern den zwei Variablen für die Unter- und Obergrenze zugewiesen.
- Dann folgt mit „for ...“ eine Schleife, die über alle Zeilen des Arbeitsblatts mit Nummern beginnend mit dem Inhalt der Variablen *von* bis zu der mit dem Inhalt der Variablen *bis* läuft: Diese Schleife beinhaltet alle Anweisungen bis zur Anweisung `next`, die die for-Schleife abschließt.
- In dieser Schleife werden nun drei Objekt-Variablen mit drei Zellen der jeweiligen Zeile verknüpft, nämlich cKonto und cBLZ, die wir für die Eingabe der IBAN-Ermittlung benötigen, und cIBAN, in die dann, wenn alles geklappt hat, die ermittelte IBAN eingetragen wird. Die Adresse einer Zelle wird dazu aus dem fixen Buchstaben für die jeweilige Spalte und der Variablen i für die Zeile zusammengesetzt.
- Eine Zelle eines Calc-Dokuments hat für ihren Inhalt die drei Darstellungsformen String, also reiner Text, Value, also Zahlwert, und Formel, aus denen man den für die jeweilige Aktion sinnvollen auswählen kann. Wir führen nun die folgenden Anweisungen nur dann aus, wenn die beiden Eingabezellen überhaupt einen Inhalt haben.

- Nun erfolgt der Aufruf des Makros *FuncErmittleIBAN*, wie wir ihn bereits aus „Beispiel 3: IBAN ermitteln mit Prüfungen, Unterprogrammaufruf“ auf Seite 20 kennen, allerdings mit anderen Variablen als Parameter. Hier wird für die Eingabe die Eigenschaft *string* verwendet. Eine Zeichenkette als Parameter hatten wir bisher so eingesetzt, weil wir eine Numerischprüfung benutzten, und das ist auch hier von Vorteil, weil in Calc bei einem nichtnumerischen Inhalt der Zelle die Eigenschaft *Value* einfach 0 enthält, also nicht verwendbar ist. Ein numerisch übergebener Parameter würde in das erforderliche Format string umgewandelt, alle nichtnumerischen Inhalte würden damit wegen der Übergabe von 0 zwar glatt durchgehen, aber nichtssagende Ergebnisse liefen. Nur bei der Übergabe als Zeichenkette wirken unsere Prüfungen.
- Wie in „Beispiel 3: IBAN ermitteln mit Prüfungen, Unterprogrammaufruf“ auf Seite 20 fragen wir das Ergebnis des Aufrufs von *FuncErmittleIBAN* ab, allerdings nun direkt, ohne eine Zwischenvariable zu verwenden. Wenn das Programm eine erfolgreiche Durchführung signalisiert, also „True“ geliefert hat, dann füllen wir die Zelle, die wir mit der Variablen *cIBAN* verknüpft haben, über die Eigenschaft *string* mit der Zeichenkette IBAN. Da diese Zeichenkette Buchstaben enthält – das Länderkürzel – , ist das auch der richtige Zugang zum Inhalt der Zelle.

Ausführung des Makros

Wenn wir das Makro nun starten (s. „Ein Makro ausführen“ auf Seite 19), dann werden wir mit dem von uns eingerichteten Dialog aufgefordert, die Unter- und Obergrenze anzugeben, Abbildung 19 zeigt den Dialog mit bereits eingegebenen Werten.

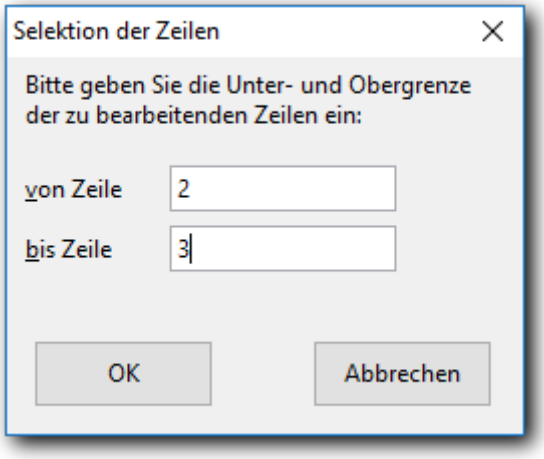

Abbildung 19: Dialog DlgSel bei der Ausführung

Wenn Sie OK betätigt haben, wird der Rest des Makros durchlaufen, im Beispielfall ist das Ergebnis eine ausgefüllte Tabelle wie in Abbildung 20.

	A	B	C
1	konto	BLZ	IBAN
2	123456	74297531	DE78742975310000123456
3	1122334455	76543210	DE31765432101122334455

Abbildung 20: Calc-Tabelle mit eingefügter IBAN

Wenn allerdings in einer Zeile ein Problem auftritt, dann wird im Unterprogramm *FuncErmittleIBAN* die Meldung ausgegeben, das Hauptprogramm *setIBAN* bricht anschließend ab. Dann müssen Sie die Fehlerursache korrigieren und an der Stelle, wo das Problem auftrat, wieder ansetzen. Da

ist es nun von Vorteil, dass unser Programm den Start bei einer beliebigen Zeile ermöglicht: Sie geben beim nächsten Aufruf die Zeile, in der Sie den fehlerhaften Inhalt korrigiert haben, als neuen Startwert an.

Resümee

Man kann mit wenig Schreibaufwand Programme erstellen, die viel erledigen.

Allerdings muss man mit den notwendigen Objekten und deren Eigenschaften und Methoden vertraut sein. Das ist nun keine triviale Aufgabe, und daher kommt auch die oft gehörte Aussage, dass die Lernkurve für das Programmieren mit LibreOffice am Anfang steil sei. Andererseits müssen Sie immer nur die Objekte verstehen, mit denen Sie gerade zu tun haben, und LibreOffice bietet über die Methoden *dbg_properties*, *dbg_methods* und *dbg_supportedInterfaces* bereits Basis-Informationen über Eigenschaften und Methoden eines Objekts an. Wesentlich verbessert aufbereitet wird diese Information durch Werkzeuge wie Xray (s. „Weitere Schritte“ auf Seite 36), die für ein Objekt alle Eigenschaften, Methoden usw. mit Typ, Parametern usw. auflisten und auch einen wiederholten Drill-Down auf einzelne komplexe Eigenschaften erlauben.

Mithilfe solcher Werkzeuge kann man, selbst wenn die Dokumentationen, wie sie in „Weitere Schritte“ auf Seite 36 angesprochen werden, für die eigene Fragestellung keine Rezepte anbieten, selbständig die infrage kommenden Objekte erforschen und daraus neue Programme entwickeln.

Weitere Einsatzmöglichkeiten für die IBAN-Ermitllung

Sie könnten nun ein Makro schreiben, das über einen Dialog die Eingabe von BLZ und Kontonummer erhält und die IBAN ausgibt. Die Werkzeuge dazu haben Sie prinzipiell alle schon kennengelernt. Dieses Makro könnten Sie einer Schaltfläche in einem Formular zuweisen. Damit hätten Sie eine Situation ähnlich den Websites der Banken und Sparkassen, die bei Überweisungen noch vielfach die Eingabe von BLZ und Kontonummer über einen Dialog als Alternative zur IBAN anbieten.

Makros aufzeichnen

Es gibt noch eine weitere Möglichkeit, Makros zu erstellen, nämlich die Aufzeichnung mit Hilfe des Makrorecorders.

Der Makrorecorder

Der Makrorecorder zeichnet die Aktionen auf, die Sie über Tastatur, Menüs oder Symbolleisten auslösen. Er kann also nur Folgen von Aktionen, die Sie in identischer Form mehrfach benötigen, speichern und auf Abruf wiederholen, aber ohne jegliche Variation oder Eingriffsmöglichkeit ihrerseits.

Es sind noch weitere Einschränkungen zu beachten:

- Manche Befehle werden nicht vollständig aufgezeichnet, z.B. wird beim Einfügen eines Rechtecks oder eines anderen grafischen Objekts in ein Writer-Dokument zwar der Befehl erfasst, aber da die Information über das Objekt nicht vorliegt, ist der Befehl unvollständig und wird daher nur als Kommentar eingefügt.
- Nicht alle Aktionen, die einen Dialog für weitere Angaben eröffnen, werden korrekt aufgezeichnet. So kann man zwar das Setzen einer Fußnote mitsamt den dabei angebotenen Wahlmöglichkeiten aufzeichnen, aber nicht die folgende Änderung, die genau die gleichen Wahlmöglichkeiten bietet; auch hier ist wieder nur der nackte Befehl ohne Parameter als Kommentar eingefügt.
- Der Makrorecorder existiert nicht in Draw, Impress und Math.

Man muss daher immer erst prüfen, ob man eine Aufgabe auch wirklich mit der Aufzeichnung als Makro lösen kann. Vieles, was man mit aufgezeichneten Makros erreichen kann, lässt sich auch auf anderem Weg, z.B. durch Kopieren in die Zwischenablage und Einfügen, die Formatübertragung oder das Suchen und Ersetzen recht gut lösen.

Da der Makrorecorder nur eingeschränkt einsetzbar ist, ist er auch standardmäßig inaktiv. Sie können ihn über **Extras → Anpassen** aktivieren, s. Kapitel 13, LibreOffice Einstellungen dieses Handbuchs.

Ein Makro aufzeichnen

Um ein Beispiel für eine Aufzeichnung zu zeigen, das sich nicht ohne weiteres durch andere Standard-Befehle (aber natürlich auch durch Programmierung, s. „Beispiel 5: Ersatz des aufgezeichneten Makros durch ein programmiertes“ auf Seite 34) lösen lässt, verwenden wir noch einmal das Thema der IBAN: Der besseren Lesbarkeit halber wird die IBAN zur Anzeige häufig nach jeweils vier Stellen durch ein Leerzeichen getrennt. Das wollen wir nun für IBANs, die in einem Text vorkommen, durch ein aufgezeichnetes Makro ausführen lassen. (Bei IBANs, die wir mit dem vorgestellten Programm ermitteln, könnten wir das ja bereits leicht im Programm tun.)

Um dieses Makro aufzuzeichnen, gehen Sie wie folgt vor:

- Sie brauchen in ihrem Text ein „Wort“, das so wie eine IBAN aufgebaut ist, also (mindestens) 22 Stellen hat. Stellen Sie den Cursor irgendwo in das Wort. Wir gestalten das Makro so, dass der Cursor irgendwo in der IBAN stehen darf, bloß nicht ganz zu Beginn.
- Wählen Sie **Extras → Makros → Makro aufzeichnen**
- es erscheint ein „schwebendes“ Fenster, mit dem Sie, wenn Sie mit den Eingaben fertig sind, die Aufzeichnung beenden können (Abbildung 21).

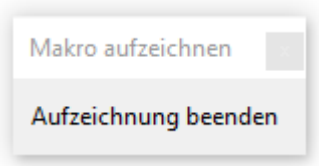

Abbildung 21: Schwebendes Fenster für Makroaufzeichnung

- Wenn dieses Fenster den Text, mit dem Sie das Makro aufzeichnen wollen, überdeckt und Sie somit nicht sehen, was Sie tun, dann können Sie das Fenster einfach an eine Stelle verschieben, wo es nicht stört (deshalb heißt es „schwebend“). Sie müssen dann nur den Fokus, der nun auf dem schwebenden Fenster ist, z.B. durch Klicken, wieder in die IBAN im Dokument setzen, mit der Sie das Makro aufzeichnen wollen.
- Um nun ein Makro aufzuzeichnen, das unabhängig davon funktioniert, wo der Cursor im Wort steht, bedienen wir uns der Tatsache, dass es eine Tastenkombination gibt, die den Cursor zum Anfang des Worts bewegt: *Strg+Nach links*.
  Weil dieser Befehl aber, falls der Cursor schon am Beginn des Wortes steht, diesen zum vorigen Wort bewegt, wurde oben darauf hingewiesen, dass er bei der Ausführung – und also auch bei der Aufzeichnung – eben nicht am Wortanfang stehen darf.
  Nachdem also unser Cursor wie vorgeschrieben irgendwo im Wort stand, wird er durch den Befehl an den Anfang des Worts gestellt.
- Betätigen Sie nun viermal die Taste *Nach rechts* und danach die *Leertaste*: nach vier Zeichen ist ein Leerzeichen eingefügt.
- Wiederholen Sie das noch viermal: Sie haben die IBAN in Vierergruppen aufgeteilt, zuletzt stehen zwei Zeichen.

- Beenden Sie die Makroaufzeichnung durch Anklicken des Befehls im schwebenden Fenster
- Es wird nun ein Fenster geöffnet, in dem Sie das Makro speichern können (Abbildung 22).

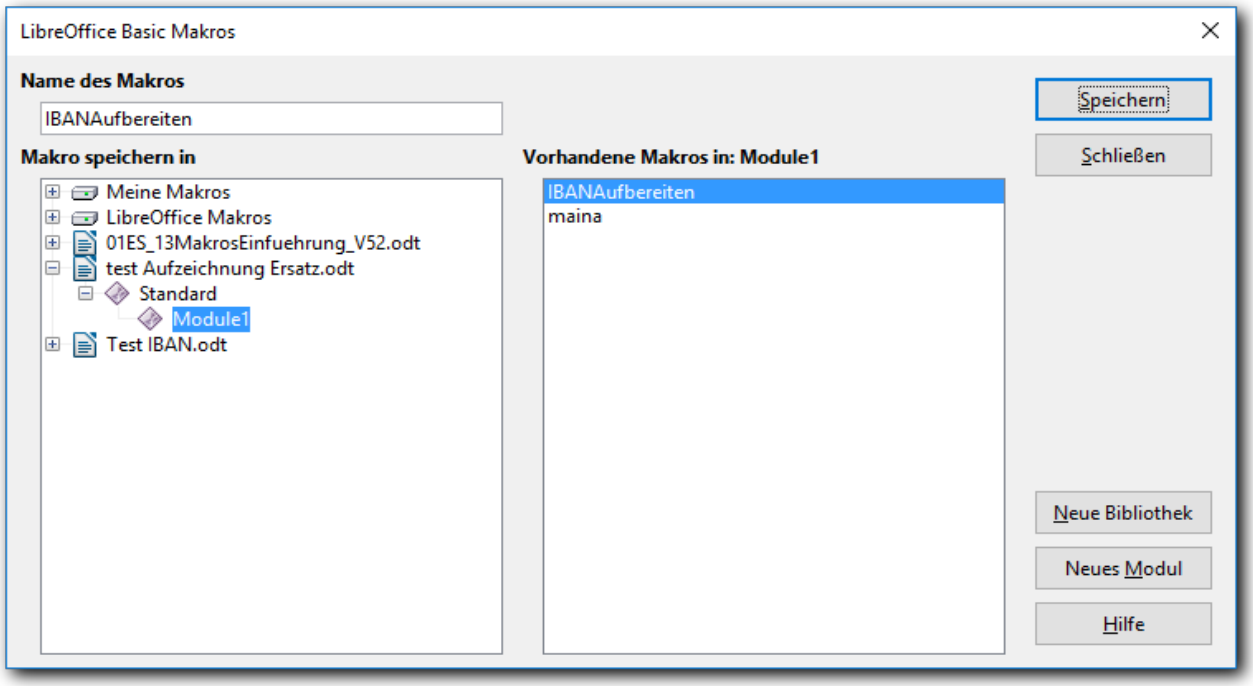

Abbildung 22: Speichern eines aufgezeichneten Makros

- Je nachdem, wo Sie das Makro speichern wollen, klappen Sie im linken Baum den richtigen Modul auf. Gegebenenfalls müssen sie mit *Neues Modul* einen neuen Modul anlegen. Sind schon Makros in dem Modul vorhanden, dann wird der erste Im Feld „Name des Makros“ vorgeschlagen, dieser Name ist auch im rechten Listenfeld markiert . Wählen Sie gegebenenfalls einen anderen Namen – damit würde dann das Makro mit diesem Namen nach einer Sicherheitsabfrage überschrieben – oder vergeben Sie einen neuen Namen, um ein neues Makro in diesem Modul zu erzeugen.
- Zum Ausführen dieses Makros müssen Sie nun nur noch den Cursor in eine weitere IBAN in Ihrem Text setzen und es ausführen; die Möglichkeiten zur Ausführung sind in „Ein Makro ausführen“ auf Seite 19 beschrieben.

Das aufgezeichnete Makro

Der Makrorecorder erstellt die Aufzeichnung, indem er die Befehle des Dispatchers abfängt und speichert. Das Dispatch Framework ist die Schnittstelle zwischen der Benutzeroberfläche und dem Kern von LibreOffice, Befehle über Tastatur, Menü oder Symbolleisten werden in einen Dispatcher-Befehl umgesetzt (z.B. Speichern eines Dokuments: die Menüwahl **Datei → Speichern**, das Symbol „Speichern“ in der Symbolleiste „Standard“ und die Tastenkombination *Strg+S* werden alle in den gleichen Dispatcherbefehl umgesetzt).

Der Makrorecorder speichert eine Abfolge dieser Dispatcherbefehle und bildet somit immer Aktionen auf der Benutzeroberfläche ab. Im Gegensatz dazu arbeiten die Programme, die mit Basic und UNO arbeiten, wie sie in den vorigen Abschnitten vorgestellt wurden, primär direkt auf den Dokumenten und greifen nur bei Bedarf auch auf Informationen der Benutzeroberfläche zu.

Das Makro können Sie wie oben bei den anderen Beispielen dargestellt in der IDE ansehen und auch verändern. Es enthält nun für jeden Tastaturbefehl einen Block mit dem Setzen einiger Parameter und dem anschließenden Aufruf des Dispatcher-Befehls. Sie erinnern sich, dass wir

fünfmal den Cursor viermal nach rechts bewegt und dann ein Leerzeichen eingegeben haben. All diese Schritte finden sich nun im Makro, für die Abbildung hier wurde es daher verkürzt. Die wiederholten Schritte – fünfmal das Einfügen eines Leerzeichens und 4 * 5 = 20 mal das Bewegen des Cursors um eine Stelle nach rechts – wurden nur angedeutet. Beachten Sie, dass dabei jedes Mal (insgesamt 26-mal) eine neue Variable *args...* definiert wird und alle zugehörigen Werte zugewiesen werden, obwohl man eigentlich nur zwei verschiedene Variable benötigen würde.

Die „Variable“ ist übrigens, genau betrachtet, eine Aufzählung, nummeriert von 0 bis 1 – das besagt das „(1)“ in der Definition – , von UNO-Objekten vom Typ „PropertyValue, die jeweils ein Wertepaar definieren, wobei der erste Wert („Name“) jeweils ein String und der zweite (“Value“) der zugehörige Wert ist. Dieser Typ von Objekten wird in LibreOffice sehr häufig benutzt, um Parameter auf eine universelle Art weiterzugeben.

```
sub Main
rem ----------------------------------------------------------------------
rem define variables
dim document   as object
dim dispatcher as object
rem ----------------------------------------------------------------------
rem get access to the document
document   = ThisComponent.CurrentController.Frame
dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")

rem ----------------------------------------------------------------------
dispatcher.executeDispatch(document, ".uno:GoToPrevWord", "", 0,
   ►Array())

rem ----------------------------------------------------------------------
dim args2(1) as new com.sun.star.beans.PropertyValue
args2(0).Name = "Count"
args2(0).Value = 1
args2(1).Name = "Select"
args2(1).Value = false

dispatcher.executeDispatch(document, ".uno:GoRight", "", 0, args2())

rem ----------------------------------------------------------------------
dim args3(1) as new com.sun.star.beans.PropertyValue
args3(0).Name = "Count"
args3(0).Value = 1
args3(1).Name = "Select"
args3(1).Value = false

dispatcher.executeDispatch(document, ".uno:GoRight", "", 0, args3())

rem ----------------------------------------------------------------------
```

hier folgen nun noch zweimal die entsprechenden Zeilen für die dritte bis vierte Auslösung der Taste *Nach rechts*. Dann folgt das Einfügen des Leerzeichens:

```
dim args6(0) as new com.sun.star.beans.PropertyValue
args6(0).Name = "Text"
args6(0).Value = " "

dispatcher.executeDispatch(document, ".uno:InsertText", "", 0,
args6())
```

hier folgen nun noch viermal die gleichen Anweisungen: viermal *Nach rechts* und *Leertaste*

```
end sub
```

Der originale Code ist also fünf- bis sechsmal so lang!

Zu Beginn wurde für den Ausdruck eine Code-Zeile, die bei dem hier verwendeten Format nicht auf eine Druckzeile passt, in zwei Zeilen umgebrochen, das wurde durch das Einfügen von ► bei der zweiten Zeilenhälfte angedeutet.

Nach den dim-Anweisungen folgen die Dispatcher Befehle. Sie sehen, dass Sie alle den gleichen Aufbau haben: Setzen der Parameter über ein Objekt des Typs *com.sun.star.beans.PropertyValue* und Aufruf des Befehls. Der erste Befehl – .uno:GoToPrevWord – benötigt keine Parameter, daher wird mit der Funktion array, die indizierte Bereiche nach Angabe in der Klammer liefert, ein leerer Bereich mitgegeben.

Würde man das selbst programmieren, würde man gleich merken, dass der zweite bis vierte Block bis auf den anderen Variablennamen args3, args4 usw. gleich aussehen wie der erste, dass also die vierfache Ausführung des dispatch-Befehls immer mit arg2 den gleichen Effekt hätte. Man könnte also durch ein bisschen Ändern das aufgezeichnete Makro schon deutlich verkürzen. Noch kürzer und auch für weitere Aufgaben zukunftsfähiger ist jedoch die Verwendung von Objekten, wie wir gleich in „Beispiel 5: Ersatz des aufgezeichneten Makros durch ein programmiertes“ auf Seite 34 sehen werden.

Aufzeichnen versus Programmieren

Wem es ausreicht, ein derart aufgezeichnetes Makro öfters zu verwenden, findet damit eine schnelle Methode, Tipparbeit zu sparen und zuverlässig Aktionsfolgen wiederholt auszuführen.

Wer jedoch weitere Logik – Reagieren auf Inhalte, Abfragen beim Benutzer usw. – einbringen will, der kommt mit dem Makrorecorder nicht weiter, sondern wird auf die Programmierung zurückgreifen müssen, zumal dort auch Objekte zur Verfügung stehen, die das Dispatcher-Modell mit seiner Einschränkung auf Oberflächen-Aktionen nicht bietet.

Beispiel 5: Ersatz des aufgezeichneten Makros durch ein programmiertes

Zum Vergleich soll die Umsetzung der gleichen Logik in einem Basic-Programm dienen, die kürzer und auch, durch die Verwendung einer Schleife und des – sicher auch laufzeitsparenden – Bewegens um vier Stellen auf einmal, besser lesbar ist. Es wurde wieder das Zeichen ► verwendet, um die Fortsetzung einer Codezeile auf der nächsten Textzeile anzudeuten.

```
Sub IBANAufbereiten

dim text as object, cursor as object, i as integer

text = ThisComponent.text
cursor = text.createTextCursorByRange
   ►(ThisComponent.currentSelection(0).Start)
cursor.goToStartOfWord(False)
for i = 1 To 5
   cursor.goRight(4, False)
   text.insertString(cursor, " ", False)
next

End Sub
```

- *ThisComponent* ist das Dokument, seine Eigenschaft *text* liefert ein sehr komplexes Objekt, das den Text mit all seinen Formatierungen, den eingebetteten Objekten wie Grafiken usw. liefert.
- Der Cursor, der in der folgenden Zeile definiert wird, arbeitet ähnlich, wie der Cursor, den Sie am Bildschirm sehen: er kann sich vorwärts und rückwärts bewegen – übrigens auch

gleich um Wörter und Paragraphen – und dabei Text auswählen – wie beim Markieren – oder auch nicht. Er wird hier bei der Definition gleich auf die Position des Anfangs des sichtbaren Cursors gesetzt; Sie erinnern sich, dass wir vorausgesetzt haben, dass der sichtbare Cursor innerhalb eines Wortes, das eine IBAN beinhalten soll, stehen soll. (Stillschweigend vorausgesetzt ist hier, dass der Bereich, den Sie selektiert haben, zusammenhängend ist, denn es wird von der currentSelection, die mehrere Teilbereiche haben kann, nur der erste (mit dem Index 0) verwendet). Wenn der sichtbare Cursor mehrere Zeichen umfasst, die Sie markiert haben, macht uns das hier zum Glück nichts aus, weil wir durch den Sprung zum Anfang des Wortes auf jeden Fall einen Cursor ohne Ausdehnung erhalten –. und das sowohl beim aufgezeichneten wie beim programmierten Makro – , aber in anderen Fällen könnte das aufgezeichnete Makro schon an seine Grenze stoßen, wenn so ein unvorhergesehener Fall auftritt. In einem programmierten Makro können Sie auf eine solche Situation reagieren, beispielsweise einen derart ausgedehnten Cursor durch *goToEnd* oder *goToStart* auf einen der Länge 0 reduzieren, wenn das für das Programm notwendig ist.

- Die nächste Zeile setzt den Cursor auf den Beginn des Wortes; der Parameter „False“ bedeutet, wie bei den folgenden Befehlen, dass der Bereich zwischen der vorigen und der neuen Cursorposition nicht markiert werden soll.
  Beachten Sie, dass dieser Befehl, im Unterschied zur Tastenkombination für das vorige Wort beim aufgezeichneten Makro, sogar funktioniert, wenn der Cursor am Beginn eines Wortes steht. Das ist zwar kein universelles Prinzip, aber doch ein Indiz, dass die Funktionen, die UNO bereitstellt, präziser und damit vielseitiger verwendbar sind als die Dispatcher-Befehle.
- Nun folgen, in einer fünffachen Schleife, das Bewegen des Cursors um 4 Stellen nach rechts (in einem Befehl!) und das Einfügen eines Leerzeichens.

Bei diesem Makro könnten Sie nun – was bei der IBAN nicht recht vorstellbar ist – Prüfungen oder Verarbeitungen der jeweils vier letzten Stellen einbauen. Das wäre in der Schleife im hier vorgestellten Programm eine einzige Stelle; im aufgezeichneten Programm müssten Sie das fünfmal tun, wobei noch nicht einmal klar ist, wie Sie dort auf die gewünschten Objekte zugreifen.

Sie sehen also, dass Sie, wenn Sie mehr als nur eine feste Abfolge von Online-Eingaben wiederholen wollen, ganz schnell beim Programmieren anlangen.

Es sollte andererseits nicht verschwiegen werden, dass es auch einige Situationen gibt, in denen Dispatcher-Befehle in eigengeschriebenen Makros verwendet werden können. Zum Beispiel gibt es für das Zurücknehmen einer Aktion keinen UNO- oder Basic-Befehl, wer so etwas in seinem Programm benötigt, greift auf das entsprechende Dispatcher-Kommando zurück.

Die Empfehlung ist also: Dispatcher bei reiner Aufzeichnung ja, ansonsten Programmierung mit Verwendung von Dispatcher in seltenen Ausnahmefällen.

Dialoge und Formulare

Einen einfachen Dialog haben Sie bereits in „Erstellen des Dialogs“ auf Seite 25 kennengelernt. Außer den gezeigten Kontrollelementen – numerische Eingabefelder und Schaltflächen – kann ein Dialog auch andere enthalten, z.B. andere Arten von Eingabefeldern, Listenfelder, Markierfelder, Optionsfelder, Baumstrukturen usw.

Ein Dialog ist ein eigenständiges Fenster, das durch eine Aktion aufgerufen wird.

Sie können Kontrollelemente aber auch in LibreOffice-Dokumenten platzieren. Dazu stehen im Wesentlichen die gleichen Kontrollelemente wie bei Dialogen zur Verfügung. Man spricht dann von Formularen, wobei der Sprachgebrauch etwas verwirrend ist:

- im technischen Sinn ist ein Formular eine Komponente eines Dokuments (ein UNO-Objekt), das optisch gar nicht erkennbar ist, sondern als Behälter für alle Kontrollelemente dient. Es wird automatisch erzeugt, wenn Sie das erste Kontrollelement einfügen. Nur die Kontrollelemente, mit denen man ja auch arbeitet, sind sichtbar.
- In Base (siehe dazu das Kapitel „Einführung in Base“ in diesem Handbuch) werden aber auch die Dokumente, die diese Kontrollelemente enthalten, als Formulare bezeichnet; ein Formular in diesem Sinn enthält also ein oder mehrere Formulare im technischen Sinn. Zu Unterscheidung könnte man daher diese Dokumente deutlicher als Formulardokumente bezeichnen.

Neben Base, wohl der Hauptanwendung für Formulare, können Sie auch in Writer, Calc, Impress oder Draw Formulare in Dokumenten erzeugen, indem Sie ein Kontrollelement einfügen. Beispiele wären unter vielen anderen:

- ein Writer-Dokument mit Eingabe-, Ankreuz- und Optionsfeldern, das wie ein Papierformular im herkömmlichen Sinn wirkt, durch die Möglichkeit, Auswahllisten mit vorgeschlagenen Einträgen zu verwenden, aber sogar noch komfortabler ist.
- Ein Calc-Dokument, in dem Sie die Berechnung erst durch eine Schaltfläche anstoßen, wenn alle notwendigen Eingaben gemacht und geprüft wurden.
- Eine Präsentation, bei der Sie dem Betrachter durch Schaltflächen die Möglichkeit geben, den Ablauf der Präsentation selbst zu variieren.

Durch die Verwendung von Dialogen und Formularen in Verbindung mit Makros haben Sie die Möglichkeit, die Standardfunktionalität von LibreOffice deutlich zu erweitern, bis hin zu Anwendungen, denen man im ersten Augenblick gar nicht ansieht, dass sie in LibreOffice ablaufen.

Extensions

Mit dem Schlagwort Extensions wird ein Konzept bezeichnet, das es ermöglicht, auf sehr einfache Weise zusätzliche Funktionalität zu einer Installation von LibreOffice hinzuzufügen. Detaillierter wird darauf im Kapitel „Anpassen von LibreOffice“ dieses Handbuchs eingegangen. Hier relevant ist, dass in den meisten Fällen auch Makrocode zur Installation hinzugefügt wird, sei es durch Basic-Code oder durch Dialoge. Ein klassisches, sehr allgemeines und nützliches Beispiel ist Xray (s. „Xray“ auf Seite 38). Sie finden den eingefügten Code in der Regel im Verzeichnis „Meine Makros & Dialoge“.

Weitere Schritte

Was Sie in dieser Einführung lesen konnten, die ja nur einen groben Überblick über dieses Gebiet geben kann, reicht natürlich noch nicht aus, um selber zu programmieren; es fehlen noch zu viele konkrete Informationen, z.B. schon zu den Befehlen von Basic.

Wer sich weiter mit Makros beschäftigen will, sollte sich zunächst die Syntax von Basic erarbeiten, was wegen des überschaubaren Umfangs der vorhandenen Befehle relativ schnell erledigt werden kann. Hilfe dazu bieten die Online-Hilfe, Bücher zu dem Thema und online verfügbare Anleitungen.

Um Programme für LibreOffice zu schreiben, benötigt man dann vor allem Kenntnisse über die Objekte von UNO. Dies ist nun ein riesiges Gebiet, und man wird sich das diesbezügliche Wissen nur schrittweise, je nach benötigten Objekten, erarbeiten. Unterstützung dazu bieten wiederum die unten genannten Bücher, die einen recht guten Überblick über die am häufigsten verwendeten Objekte liefern und damit auch gleichzeitig die Prinzipien der Arbeit mit diesen Objekten verdeutlichen. Die Online-Dokumentation des API (für den Zugriff auf UNO) ist ein technisches Nachschlagewerk, das auch als Spezifikation für die Programmierung des API dient, ist aber zum

Erlernen des Umgangs mit UNO kaum geeignet, sondern erst für den brauchbar, der sich schon einigermaßen mit dem Umgang mit diesen Objekten auskennt.

Basic

Die Dokumentation der Basic-Funktionen finden Sie in der Hilfe von LibreOffice. Rufen Sie dazu den Menüpunkt **Hilfe → LibreOffice Hilfe** auf. In der Auswahlliste auf der linken Seite doppelklicken Sie dann „Makros und Programmierung“.

Hinweis	Beachten Sie, dass die Hilfe offline nur dann zur Verfügung steht, wenn Sie das Helppack für LibreOffice zusätzlich zur Installation heruntergeladen und installiert haben.

UNO

Die – oft sehr komplexen – Objekte, mit denen Sie in LibreOffice arbeiten können, werden über die Schnittstelle UNO in Basic zur Verfügung gestellt. Die Dokumentation von UNO ist nur online verfügbar. Bedingt durch die objektbezogene Konstruktion von UNO, die dem Stand der Technik entsprechend ganz wesentlich auf der Vererbung von Eigenschaften und Methoden von übergeordneten Klassen basiert, ist diese Dokumentation zwar im Detail, nämlich den in dieser Vererbungshierarchieebene neu hinzugefügten Eigenschaften und Methoden, aussagekräftig, erfordert aber in allen geerbten Attributen und Methoden einen teilweise mehrfachen Rückgriff auf übergeordnete Objekte, von denen diese geerbt wurden. Dies liegt an dem Zweck dieser Dokumentation, nämlich eine Definition für die Realisation der Schnittstelle zu diesen Objekten für alle Programmiersprachen zu bieten.

Eine Gesamtübersicht über die für ein Objekt vorhandenen Attribute und Methoden liefert diese Hilfe konstruktionsbedingt also nicht. Diese Übersicht bieten Standardfunktionen von LibreOffice (Methoden, die mit dbg_ beginnen sowie daraus abgeleitet Funktionen in *LibreOffice Makros → Tools → Debug*) in rudimentärer sowie das Werkzeug Xray (s. „Xray“ auf Seite 38) in sehr nützlicher und anwenderfreundlicher Form.

Bücher

Das umfassende deutschsprachige Buch über die Programmierung mit Makros von Thomas Krumbein, der auch ein aktiver Teilnehmer auf den Diskussionsseiten von LibreOffice ist sowie ein aktives Mitglied in der Document Foundation, die das Projekt LibreOffice trägt, mit dem Titel „Makros in OpenOffice.org 3“ ist derzeit vergriffen. Eine Neuauflage im Tintal-Verlag ist geplant.

Wer Französisch versteht, kann auf das exzellente Buch von Bernard Marcelly & Laurent Godard, Programmation OpenOffice.org et LibreOffice zurückgreifen. Es ist, gesamt gesehen, umfassender als Krumbein, allerdings hat Krumbein auch einige Punkte, die bei den beiden nicht vorkommen.

Im englischsprachigen Bereich ist Andrew Pitnoyiak einer der Pioniere der Makroprogrammierung. Von ihm gibt es ein nach wie vor erhältliches Buch „OpenOffice.org Macros Explained“, von dem er aber seit langem eine Aktualisierung in Arbeit hat, die er in Teilen auch veröffentlicht hat (http://www.pitonyak.org/book/). Außerdem hat er ein Dokument zusammengestellt, das keine systematische Einführung darstellt, dafür aber eine Fülle von Lösungen zu konkreten Einzelfragen liefert (http://pitonyak.org/oo.php).

Die englische Version dieses Handbuchkapitels nennt außerdem die folgenden Bücher, die sich mit speziellen Aspekten der Programmierung beschäftigen:

Dr. Mark Alexander Bain’s *Learn OpenOffice.org Spreadsheet Macro Programming*. *http://www.packtpub.com/openoffice-ooobasic-calc-automation/book*.

Roberto Benitez's *Database Programming with OpenOffice.org Base & Basic*.

http://www.lulu.com/product/paperback/database-programming-with-openofficeorg-base-basic/3568728

Xray

Xray ist eine Extension, die das Arbeiten mit Makros erleichtert. Weitergehende Informationen zum Thema „Extension“ finden Sie im Kapitel „Anpassen von LibreOffice“ des Handbuchs „Erste Schritte“. Xray wird als Makrocode zu Ihrer LibreOffice-Installation hinzufügt und mittels eines Funktionsaufrufs oder auch – bei vorheriger entsprechender Einrichtung durch Sie – durch eine Aktion Ihrerseits ausgeführt. Nach dem Aufruf werden zu dem jeweils ausgewählten Objekt alle Informationen über „Eigenschaften“ – auch komplexe, mit dem Objekt verknüpfte Objekte zählen dazu – und Methoden usw. angezeigt.

Xray verwendet die von LibreOffice bereitgestellten Basisfunktionen. Xray bereitet deren Ergebnisse aber durch Verknüpfungen von Informationen wesentlich besser zugänglich auf und ist somit ein hervorragendes Mittel, um selber die Möglichkeiten zu erforschen, die LibreOffice mit seinen UNO-Objekten bietet. Xray ist, da es vollständig mit Basic programmiert ist, auch ein wunderbares Beispiel, was man alles mit Basic erreichen kann.

Xray können Sie von der Seite von Bernard Marcelly, der auch einer der Autoren von „Programmation OpenOffice.org et LibreOffice“ ist, herunterladen und installieren. Die englischsprachige Seite erreichen Sie mit http://berma.pagesperso-orange.fr/index2.html, dort finden Sie auch eine deutschsprachige Version von Xray.

Online-Ressourcen

https://wiki.documentfoundation.org/Documentation/Other_Documentation_and_Resources#Programmers. Der Link führt direkt zum Abschnitt über Programmierung; die ganze Seite ist darüber hinaus eine Fundgrube für Dokumentationen.

https://wiki.documentfoundation.org/Macros Die Seite führt ebenfalls zu verschiedenen Dokumentationen, teilweise zu den gleichen, und auch zu den Seiten über das LibreOffice API. Die deutschsprachige Seite *http://wiki.documentfoundation.org/Macros/de* hat einen etwas anderen Inhalt, führt aber z.B. zu einer Kurzanleitung zur Programmierung von Makros

Da die Seite des LibreOffice API von großer Wichtigkeit ist, wenn man Attribute oder Methoden gefunden hat (beispielsweise über Xray), aber nicht genau weiß, was sie bedeuten, sei hier auch noch der direkte Link genannt: *http://api.libreoffice.org/docs/idl/ref/index.html*.

Hinweis	Aufgrund der Entwicklung bei OpenOffice.org können wir nicht garantieren, dass die darauf bezogenen Links auch in Zukunft in dieser Form zur Verfügung stehen.

In LibreOffice enthaltene Makros

Viele gute Makros sind schon in LibreOffice enthalten. Verwenden Sie **Extras** → **Makros** → **Makros verwalten** → **LibreOffice Basic...**, um den Dialog *LibreOffice Basic Makros* zu öffnen. Klappen Sie den Speicherort „LibreOfficeMakros“ auf. Er enthält zum Teil Bibliotheken, die von LibreOffice genutzt werden und die recht komplex sein können. Die Bibliothek „Tools“ jedoch ist dafür gedacht, kleine, häufig verwendbare Funktionen zur Verfügung zu stellen, die Sie in Ihren Programmen aufrufen können. Diese Funktionen ersparen einerseits Schreibarbeit, andererseits bieten sie Sicherheit, weil sie von Kennern erstellt und ausgetestet wurden. Die Funktionen in dieser Bibliothek sind häufig recht kurz und daher gut geeignet, sie zu analysieren und dadurch Basic besser kennenzulernen.