

**LibreOffice**  
The Document Foundation

Erste Schritte

## *Kapitel 13*

### *Einführung in Makros*

*Immer wiederkehrende Schritte automatisieren*

## Copyright

---

Dieses Dokument unterliegt dem Copyright © 2017. Die Beitragenden sind unten aufgeführt. Sie dürfen dieses Dokument unter den Bedingungen der GNU General Public License (<http://www.gnu.org/licenses/gpl.html>), Version 3 oder höher, oder der Creative Commons Attribution License (<http://creativecommons.org/licenses/by/3.0/>), Version 3.0 oder höher, verändern und/oder weitergeben.

Warennamen werden ohne Gewährleistung der freien Verwendbarkeit benutzt.

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

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

## Mitwirkende/Autoren

Gerhard Weydt

## Rückmeldung (Feedback)

Kommentare oder Vorschläge zu diesem Dokument können Sie in deutscher Sprache an die Adresse [discuss@de.libreoffice.org](mailto: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 13.10.2019 Basierend auf der LibreOffice Version 6.2.

# Inhalt

---

Einleitung .....	5
Was sind Makros und Dialoge? .....	5
Makros .....	5
Dialoge .....	6
Makros ausführen und speichern .....	6
Organisation der Makros .....	6
Bibliotheken .....	7
Makros ausführen .....	7
Makro direkt ausführen .....	7
Makro über die Makroverwaltung ausführen .....	8
Makro durch Tastaturbefehle oder Symbole ausführen .....	8
Makros verwalten .....	8
Aufruf der Makroverwaltung .....	8
Verwalten von Makros .....	9
Wo sollen Makros gespeichert werden? .....	9
Erstellen einer Bibliothek .....	10
Erstellen eines Moduls .....	11
Makros aufzeichnen .....	12
Der Makrorecorder .....	12
Ein Makro aufzeichnen .....	13
Das aufgezeichnete Makro .....	14
Schnelleres Ausführen des Makros .....	15
Ein komplizierteres Beispiel .....	15
Das aufgezeichnete Makro .....	16
Programmieren von Makros mit Basic .....	17
Hinweise bzgl. der Makro-Beispiele in diesem Handbuch .....	17
Basic .....	17
Objekte von LibreOffice .....	18
Erstellen von Makros .....	19
Makros bearbeiten .....	19
Die Basic-IDE .....	19
Beispiel 1: IBAN berechnen .....	20
Aufgabenstellung .....	20
Das erste Makro, mit einem kleinen Problem .....	21
Finden der Fehlerursache .....	23
Beispiel 2: Das korrigierte Makro .....	24
Beispiel 3: IBAN ermitteln mit Prüfungen, Unterprogrammaufruf .....	26
Makros im produktiven Einsatz .....	30
Beispiel 4: Massensetzung der IBAN in einer Calc-Tabelle .....	30
Aufgabenstellung .....	31
Anlegen des Makros .....	31
Erstellen des Dialogs .....	31
Arbeiten mit UNO-Objekten .....	33
Das Makro .....	34
Ausführung des Makros .....	36
Resümee .....	36

<i>Weitere Einsatzmöglichkeiten für die IBAN-Ermittlung</i> .....	37
<i>Aufzeichnen versus Programmieren</i> .....	37
<i>Beispiel 5: Ersatz des aufgezeichneten IBAN-Makros durch ein programmiertes</i> .....	37
<i>Dialoge und Formulare</i> .....	38
<i>Extensions</i> .....	39
<i>Weitere Schritte</i> .....	39
<i>Basic</i> .....	40
<i>UNO</i> .....	40
<i>Bücher</i> .....	40
<i>Xray</i> .....	41
<i>Online-Ressourcen</i> .....	41
<i>In LibreOffice enthaltene Makros</i> .....	41

## Einleitung

---

In diesem Kapitel werden zwei deutlich unterschiedliche Methoden beschrieben, mit denen Sie wiederholt benötigte Befehlsfolgen abspeichern und somit bei Bedarf immer wieder ausführen können, was Ihnen bei der erneuten Verwendung erhebliche Tipp-, Klick- und Denkarbeit ersparen kann.

Da das Ergebnis beider Methoden in der gleichen Form (als Makro) gespeichert und damit auch verwendet wird und sich beide Methoden auch kombinieren lassen, ist es trotz der Unterschiede sinnvoll, sie beide in einem gemeinsamen Kapitel zu beschreiben.

Die beiden Methoden sind:

1. das Aufzeichnen von Benutzereingaben durch den Makrorecorder, s. „Makros aufzeichnen“ auf Seite 12. Das ist ohne zusätzliches Wissen möglich, allerdings sind die gebotenen Möglichkeiten eingeschränkt.
2. das eigene Erstellen von Makros, s. „Erstellen von Makros“ auf Seite 19. Hier haben Sie weitaus mehr Möglichkeiten, z.B. was den Zugriff auf die Objekte von LibreOffice betrifft, aber vor allem auch durch die Möglichkeit, Verarbeitungslogik zu definieren, indem Sie auf Dateninhalte reagieren können und abhängig davon verschiedene Aktionen ausführen können. Allerdings müssen Sie dazu eine der von LibreOffice unterstützten Programmiersprachen verwenden, was zunächst eine gewisse Hürde darstellt.

Sie können Makros auch von außen erhalten und in Ihrem System abspeichern (s. Verwalten von Makros“ auf Seite 9); die Verwendung erfolgt auf die gleiche Weise wie bei den von Ihnen erzeugten Makros.

### 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.

### Hinweis

Auch Erweiterungen (s. Kapitel 14 – „LibreOffice anpassen“) enthalten in der Mehrzahl Makros, aber da bei diesen in der Regel die Befehlsfolgen bequem durch Menüpunkte oder Symbole ausgeführt werden und Sie daher gar nicht mit den Makros selbst in Berührung kommen, werden sie nicht in diesem Kapitel behandelt.

Dieses Kapitel bietet ausschließlich Beispiele in der Sprache Basic an, die in LibreOffice am leichtesten einsetzbar ist, weil sie ohne zusätzlichen Installationsaufwand einsetzbar und auch am besten unterstützt und dokumentiert ist.

Was sind Makros und Dialoge?

Die sogenannte Makroverwaltung behandelt neben den eigentlichen Makros auch Dialoge. Diese beiden Klassen werden hier kurz beschrieben, das Wesentliche wird am besten an den konkreten Beispielen weiter unten in diesem Kapitel oder bei der aktiven Beschäftigung mit ihnen klar.

## Makros

Makros sind, wie schon gesagt, Folgen von Befehlen oder Anweisungen, die als Gesamtheit gespeichert werden, damit sie wiederholt ausgeführt werden können, sei es, dass man mehrmals – regelmäßig oder direkt hintereinander – die gleiche Aktion mit unterschiedlichen Daten ausführen muss, sei es in verschiedenen Umgebungen, die die gleiche Aufgabenstellung benötigen. Im

ersten Fall kommen Sie vielleicht mit einem aufgezeichneten Makro aus, im zweiten werden Sie bereits einige Kenntnisse der Programmierung benötigen.

Für den zweiten Fall bieten die Bibliotheken unter „LibreOffice Makros“ (s. „Organisation der Makros“ auf Seite 6) viele Beispiele, vor allem in „Misc“ und „Tools“ sind viele in ihrer Funktion recht leicht verständliche zu finden; als ein Beispiel sei die Funktion Round genannt, die einen ganzzahligen oder dezimalen Wert auf eine gegebene Anzahl von Vorkommastellen rundet, eine Funktion, die man in vielen Programmen gebrauchen kann.

Makros sind im Wesentlichen das gleiche wie Programme bzw. Unterprogramme, der Sprachgebrauch hängt unter anderem davon ab, wie und zu welchem Zeitpunkt der in der jeweiligen Programmiersprache geschriebene Text in Maschinencode umgesetzt wird.

## Dialoge

Da Dialoge in der Makroverwaltung integriert sind, sollen sie kurz angesprochen werden, auch wenn sie im Rahmen dieser Einführung nur an einem Beispiel gestreift werden können.

Ein Dialog ist ein neues, meist eher kleines, Fenster, das aufgrund einer Aktion Ihrerseits auf dem Bildschirm geöffnet wird und von Ihnen eine Eingabe, meist in Form von einzugebenden Daten, manchmal aber auch nur zum Beantworten einer Frage durch Ja oder nein o.ä., erwartet, die dann im weiteren Verlauf des den Dialog erzeugenden Makros benötigt wird; wenn die Eingabe erfolgt ist, verschwindet das Fenster in der Regel. Etwas ähnliches, noch recht einfaches erzeugt z.B. der Menübefehl **Bearbeiten** → **Suchen und Ersetzen...**, die meisten Dialoge, die von LibreOffice verwendet werden, sind recht kompliziert (wären aber meist prinzipiell auch mit den in der Makroverwaltung angebotenen Mitteln realisierbar).

Ganz einfache Dialoge, die neben der Darstellung einiger Informationen nur Antworten auf eine einzelne Frage durch z.B. Ja, Nein, OK, Abbrechen, Wiederholen, Ignorieren oder die Eingabe eines einzigen numerischen textlichen Werts ermöglichen, können durch bereits vorhandene Funktionen erzeugt werden, komplizierte entwerfen Sie in einer grafischen Oberfläche selbst; ein einfaches Beispiel findet sich in „Erstellen des Dialogs“ auf Seite 32.

Makros ausführen und speichern

Bevor wir Ihnen zeigen, wie Sie selbst Makros erstellen können, wollen wir zunächst darstellen, wie Sie eigene oder von anderswoher erhaltene Makros ausführen und ggf. speichern können.

## Organisation der Makros

Makros werden in einer hierarchischen Struktur abgelegt. Zur obersten, von LibreOffice vorgegebenen Ebene erfahren Sie gleich unten mehr. Die nächste Ebene ist die der Bibliotheken, diese enthalten jeweils ein oder mehrere Module, und ein Modul kann ein oder mehrere Makros enthalten.

Rufen Sie **Extras** → **Makros** → **Makros ausführen** auf. Sie erhalten einen Dialog, in dessen linken oberem Teil die Struktur der Makro-Bibliotheken ähnlich wie in Abbildung 1 dargestellt wird:

- unter „Meine Makros“ werden die von Ihnen erstellten oder eingefügten, allgemeingültigen (also für alle Dateien einsetzbare) Makros abgelegt.
- Unter „LibreOffice Makros“ finden sich Makros, die mit LibreOffice ausgeliefert werden und daher auch nicht veränderbar sind. Diese Makros können Sie verwenden, wenn Sie selbst Makros programmieren wollen, eventuell werden sie auch von LibreOffice selbst oder einer Erweiterung benutzt.
- Als letztes wird für jedes geöffnete Dokument ein Eintrag angezeigt, und wenn Sie ihn bis zum Ende aufklappen, werden, sofern vorhanden, die Makros angezeigt, die in diesem Dokument gespeichert sind. Diese Makros stehen auch nur zur Verfügung, wenn das sie enthaltende Dokument geöffnet ist. (Zur Frage der Speicherung im Dokument oder übergreifend in „Meine Makros“ s. „Wo sollen Makros gespeichert werden?“ auf Seite 5).

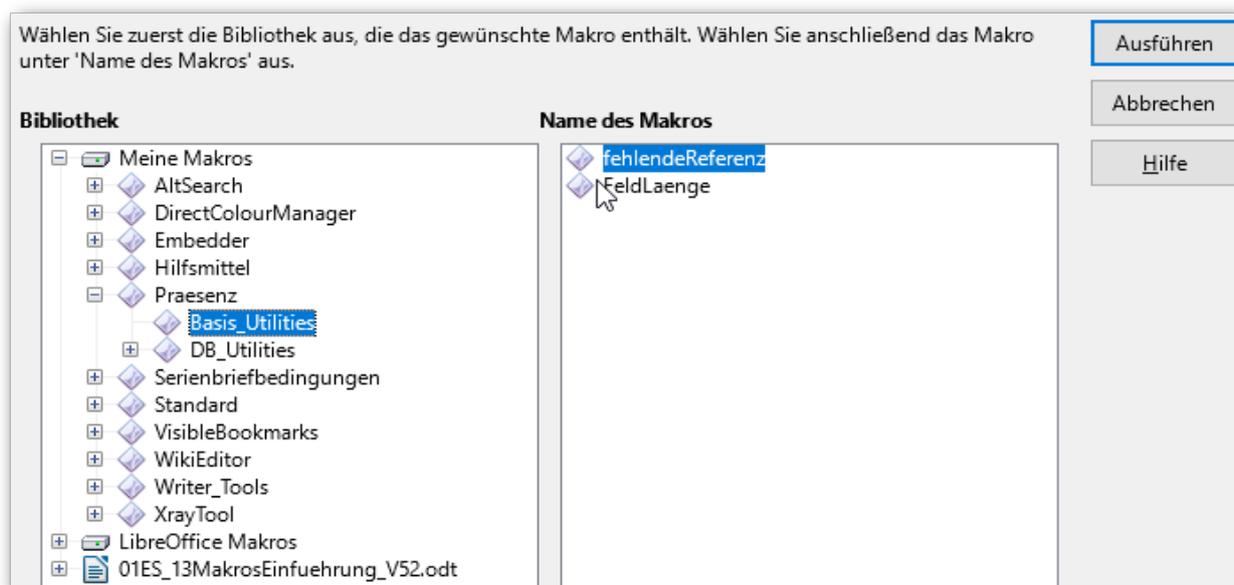


Abbildung 1: Die Makro-Bibliotheken

In Abbildung 1 sehen Sie im Bereich „Meine Makros“ die Bibliothek „Praesenz“, die zwei Module enthält, der erste, „Basis\_Utilities“, ist markiert, und daher sehen Sie rechts die beiden Makros, die er enthält.

Diese Struktur ermöglicht Ihnen, den Überblick über die gespeicherten Makros zu behalten, indem Sie entsprechende Bibliotheken und Module anlegen, wenn Sie selbst Makros erstellen. Extensions (s. Kapitel 14 „LibreOffice anpassen“ dieses Handbuchs) speichern ihre Makros auch in „Meine Makros“ und liefern üblicherweise eine eigene Bibliothek mit. Das trifft für mehr als die Hälfte der Bibliotheken in der Abbildung zu, z.B. XrayTool, das in „Xray“ auf Seite 41 erwähnt wird.

## Bibliotheken

Jede Bibliothek kann einen oder mehrere Module enthalten, und jeder dieser Module enthält ein oder mehrere Makros. Abbildung 1 zeigt zwei Module der Bibliothek „Praesenz“ und die beiden Makros des ersten Moduls. Zum Sinn dieser Gliederung mehr in „Verwalten von Makros“ auf Seite 9, wenn es um das Speichern von Makros geht.

## Makros ausführen

### Makro direkt ausführen

Wenn Sie wie in Abbildung 1 ein Makro gewählt haben, können Sie es durch das Betätigen der Schaltfläche *Ausführen* ausführen.

### Makro über die Makroverwaltung ausführen

Auch von der Makroverwaltung aus, die weiter unten beschrieben wird, kann ein Makro aufgerufen werden. Das ist von Vorteil, wenn Sie ein Makro mehrmals hintereinander benötigen, denn die Makroverwaltung ist ein eigenes Fenster und bleibt bei dem Makro geöffnet, so dass Sie es sehr schnell von dort aus wieder auslösen können. Ein Beispiel für dieses Vorgehen wird in „Schnelleres Ausführen des Makros“ auf Seite 15 beschrieben.

## **Makro durch Tastaturbefehle oder Symbole ausführen**

Sie können Makros mit Tastaturkombinationen oder Symbolen verknüpfen, damit Sie sie durch deren Betätigung direkter und schneller ausführen können. Einzelheiten dazu finden Sie in Kapitel 14 – „LibreOffice anpassen“ dieses Handbuches. Auch über die Makroverwaltung (s. „Aufruf der Makroverwaltung“ auf Seite 8) können Sie diese Verknüpfung vornehmen.

Bedenken Sie dabei jedoch, dass die Tastenbelegung für ganz LibreOffice oder zumindest für eine Komponente gültig ist. Verwenden Sie diese Möglichkeit daher nur, wenn Sie sie immer wieder benötigen, oder löschen Sie die Tastaturbelegung lieber wieder, wenn Sie sie nicht mehr brauchen.

## **Makro durch Ereignis auslösen**

Diese Möglichkeit wollen wir hier nur erwähnen, denn obwohl sie sehr bedeutsam ist, können wir sie in diesem Einführungskapitel nicht näher behandeln, da dafür tiefere Kenntnisse als die hier vermittelbaren benötigt werden. In Dialogen und Formularen (s. „Dialoge und Formulare“ auf Seite 39) gibt es für die darin enthaltenen Kontrollelemente wie Eingabefelder, Schaltflächen usw. jeweils eine Vielzahl von Ereignissen (z. B. Feld geändert, Schaltfläche betätigt, vor/nach der Datenbankänderung), durch die jeweils ein Makro aufgerufen werden kann. Dadurch kann man komplexe Anwendungen steuern und den Benutzer durch Prüfungen unterstützen. Ein knappes Beispiel wird im Abschnitt „Direkte Übernahme des Mediums (mit Makro)“ des Base-Handbuchs vorgestellt.

## **Makros verwalten**

Wenn Sie zunächst nur am Aufzeichnen von Makros interessiert sind, können Sie diesen Abschnitt eventuell überspringen, die wenigen Aktionen, die dann diesbezüglich notwendig sind, werden wir konkret beschreiben. Sobald Sie jedoch irgendetwas benötigen, was über das Aufzeichnen hinausgeht, ist dieser Abschnitt wichtig.

### **Aufruf der Makroverwaltung**

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

Wenn Sie allerdings zunächst nur bestehende Makros ändern wollen, können Sie eventuell etwas schneller und ohne vielleicht irritierende Schaltflächen zu Ihrem Ziel gelangen, wenn Sie **Extras** → **Makros** → **Makros bearbeiten** aufrufen. Letztlich landen Sie auf beiden Wegen in der gleichen Oberfläche, der Basic-IDE. Für gewisse übergreifende Aufgaben, z.B. die Verwaltung von Bibliotheken, benötigen Sie aber zunächst die allgemeine Verwaltung.

Wir beschreiben hier deshalb die allgemeine Verwaltung und weisen an der entsprechenden Stelle darauf hin, wo sich diese mit der „Bearbeitung“ trifft.

Wenn Sie also die allgemeine Makro-Verwaltung über **Extras** → **Makros** → **Makros verwalten** → **LibreOffice Basic...** aufrufen, erhalten Sie einen Dialog ähnlich Abbildung 1, mit der gleichen Darstellung der Organisation der Makros, allerdings mit zusätzlichen oder anderen Schaltflächen:

- **Ausführen:** bietet die gleiche Funktion wie „Makro direkt ausführen“ auf Seite 7
- **Schließen:** schließt das Fenster
- **Zuweisen:** ermöglicht die Zuweisung des Makros zu einer Tastenkombination oder einem Menübefehl
- **Bearbeiten:** wählt das markierte Makro zur Bearbeitung aus, genauso wie über **Extras** → **Makros** → **Makros bearbeiten**; daraufhin wird die Basic-IDE geöffnet. Deren Verwendung wird in „Die Basic-IDE“ auf Seite 19 beschrieben.
- **Löschen:** Damit löschen Sie das markierte Makro

- **Verwalten:** hier rufen Sie die Verwaltung der Makrostruktur auf, die wir in „Organisation der Makros“ auf Seite 6 bereits vorgestellt haben. Die Beschreibung finden Sie in den folgenden Abschnitten.
- **Hilfe:** hiermit können Sie, wie aus dem Standardmenü, die LibreOffice-Hilfe aufrufen.

### Verwalten von Makros

Der Aufruf der Makro-Verwaltung öffnet den neuen Dialog entsprechend Abbildung 2.

Sie können hier

- eine neue Bibliothek anlegen (s. „Erstellen einer Bibliothek“ auf Seite 11)
- in einer bestehenden Bibliothek einen neuen Modul anlegen oder einen bestehenden löschen (s. „Erstellen eines Moduls“ auf Seite 11).
- das gleiche für Dialoge ausführen. Da das im Prinzip genauso funktioniert wie für Module, gehen wir darauf nur im Rahmen des Ausblicks auf weitere Möglichkeiten in „Erstellen des Dialogs“ auf Seite 32 ein.

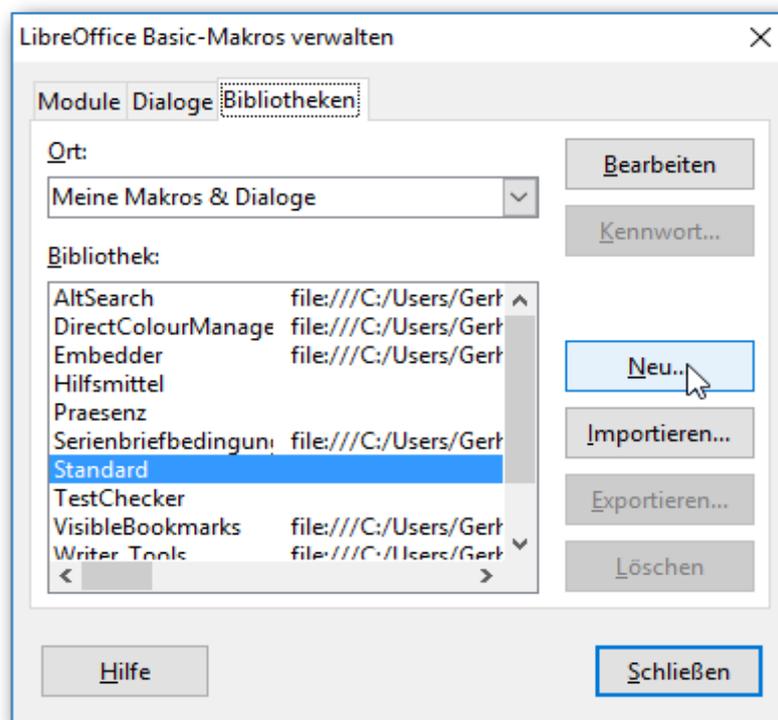


Abbildung 2: Basic-Makros verwalten - Bibliotheken

### Wo sollen Makros gespeichert werden?

Die Frage des Speicherorts ist bei den allerersten Makros, die man erstellt, insbesondere bei aufgezeichneten Makros, höchstwahrscheinlich von sehr untergeordneter Bedeutung, aber sobald man nur ein bisschen weitergearbeitet hat, stellt sie sich doch schon als relevant heraus. Deshalb ist es sinnvoll, dieses Thema bereits in dieser Einführung etwas detaillierter darzustellen, damit Sie es zumindest überfliegen und dann bei Bedarf darauf zurückgreifen können.

## Zentral oder dezentral?

In „Organisation der Makros“ auf Seite 6 wurden die drei Arten von Speicherorten für Makros beschrieben. Für das Anlegen einer neuen Bibliothek kommt allerdings die Auswahl „LibreOffice Makros & Dialoge“ nicht in Frage, die Struktur dieses Speicherorts kann nicht verändert werden, Sie können dort nur die Makros selbst verändern.

Sie haben daher die Auswahl, ein Makro, ein Modul oder gar eine ganze Bibliothek entweder in einem Dokument zu speichern oder für alle Dokumente zugänglich in „Meine Makros und Dialoge“ abzulegen. Welche Wahl zu treffen ist, ist nicht in jedem Fall leicht zu entscheiden. Makros oder die größeren genannten Einheiten, die öfters in verschiedenen Zusammenhängen benötigt werden, sind in der Regel am besten in „Meine Makros und Dialoge“ aufgehoben, da sie dann auch für neue Dokument sofort verwendbar sind und nur einmal im ganzen System vorhanden sind, was sie leichter wartbar macht.

Makros, die nur in einem Zusammenhang benötigt werden, werden in der Regel in dem Dokument abgelegt, das diesen Zusammenhang behandelt. Das hat den großen Vorteil, dass man das Dokument einfach auf einen anderen Rechner kopieren kann und dann alle notwendigen Funktionen verfügbar sind.

In komplizierteren Zusammenhängen wird man sich oft für einen Kompromiss zwischen diesen beiden Extremen entscheiden und z.B. eine gewisse Redundanz (mehrfaches Vorhandensein der gleichen Makros an mehreren Stellen, mit dem Nachteil der dadurch notwendigen parallelen Änderung und der Gefahr von nicht konsistenten Anpassungen) in Kauf nehmen. Aber selbst bei den beiden im Folgenden vorgestellten Beispielen von aufgezeichneten Makros, die ja einfache Situationen behandeln, ist nicht auf den ersten Blick eindeutig klar, wo sie am besten zu speichern sind.

Aber solange die Verteilung von Dokumenten mitsamt Makros nicht von Anfang an im Vordergrund steht, ist normalerweise die Speicherung in „Meine Makros und Dialoge“ eine gute Wahl. Dieser Bereich wird dadurch zwar größer und dadurch vielleicht schlechter überschaubar, aber das können Sie durch eine sinnvolle Gliederung in Bibliotheken und Module auffangen. Von zentralen Makros später in manchen Fällen Kopien in Dokumente einzufügen und die weitere Konsistenz dieser Kopien durch entsprechende Dokumentation sicherzustellen, ist meist weitaus einfacher als nachträglich in verschiedenen Dokumenten verstreute Makros, die das gleiche tun (sollten), in ein gemeinsames Makro zusammenzufügen.

## Wahl von Bibliotheken und Modulen

Wie viele und welche Bibliotheken und Module Sie anlegen, ist weniger problematisch. Allerdings sollten die Makros so auf die Bibliotheken und Module aufgeteilt werden, dass Sie und andere, die damit arbeiten sollen, sich auch später wieder schnell zurechtfinden können (vergeben Sie vor allem aussagekräftige Namen).

Für den Anfang können Sie erst einmal den üblichen Vorschlag „Module1“ für den Modul und „Standard“ für die Bibliothek verwenden; letzteres vor allem beim Speichern im Dokument, beim Speichern in „Meine Makros und Dialoge“ sollten Sie ziemlich bald lieber eine eigene Bibliothek wählen.

Beim Speichern in der Bibliothek „Standard“ sollten Sie, wenn Sie eine größere Anzahl von Makros haben, zurückhaltend sein, und lieber in eigenerstellten Bibliotheken speichern. Denn diese Bibliothek wird beim Start von LibreOffice bzw. beim Öffnen des sie enthaltenden Dokuments geladen, was sich auf die Ladezeit auswirkt. Andere Bibliotheken laden Sie, wenn Sie sie benötigen, erst durch einen Befehl. In größeren Umgebungen sollte also diese Bibliothek üblicherweise nur Module enthalten, die von Anfang an benötigt werden, also z.B. bei den Ereignissen „Programmstart“, „Laden des Dokuments beendet“ u.ä. aufgerufen werden.

Ein Beispiel dafür wäre das folgende Makro, das die beiden Bibliotheken „XrayTool“ und „Praesenz“, die schon in „Organisation der Makros“ auf Seite 6 angesprochen wurden, lädt, weil sie immer zur Verfügung stehen sollen.

```

Sub LoadingLibraries
  BasicLibraries.LoadLibrary("XrayTool")
  BasicLibraries.LoadLibrary("Praesenz")
End Sub

```

Auch wenn die Makrosprache Basic erst weiter unten erläutert wird, dürfte dieses kleine Stückchen Code verständlich sein.

### Erstellen einer Bibliothek

Wählen Sie den Reiter *Bibliotheken* (Abbildung 3).

In der Auswahlliste *Ort* können Sie unter den in „Organisation der Makros“ auf Seite 6 genannten Möglichkeiten auswählen. Für das Anlegen einer neuen Bibliothek kommt allerdings, wie in „Zentral oder dezentral?“ auf Seite 10 gesagt, die Auswahl „LibreOffice Makros & Dialoge“ nicht in Frage, da die Struktur dieses Teilbereichs nicht verändert werden kann.

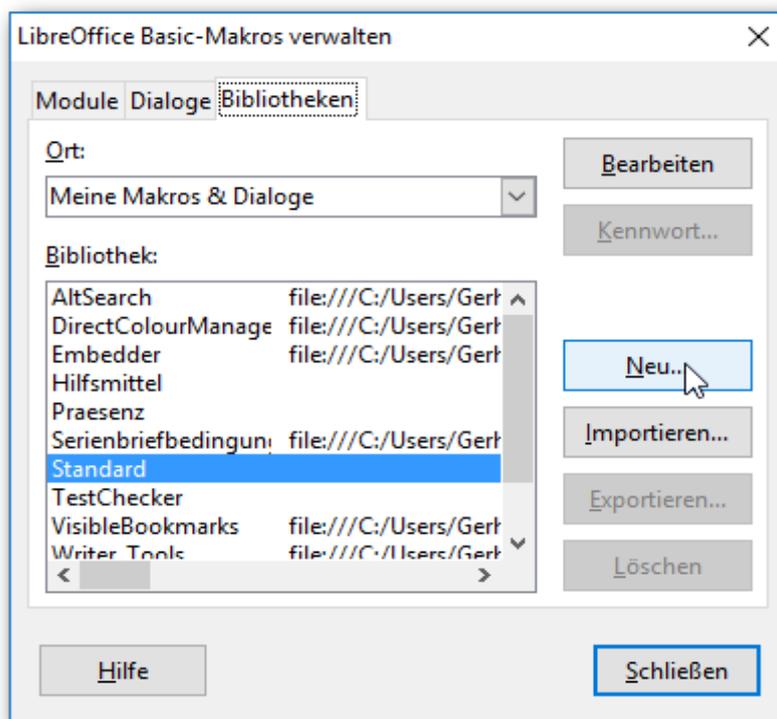


Abbildung 3: Neue Bibliothek

Wählen Sie den gewünschten Ort und betätigen Sie die Schaltfläche **Neu**. Sie werden aufgefordert, einen Namen für die neue Bibliothek zu vergeben. Nach der Vergabe und Betätigen von **OK** sehen Sie die neue Bibliothek am Ende der Liste der Bibliotheken. Bei einer späteren Anzeige der Liste der Bibliotheken ist sie dann nach dem Alphabet einsortiert.

### Erstellen eines Moduls

Wählen Sie den Reiter *Module* und ggf. dann die Bibliothek, die den neuen Modul enthalten soll. Betätigen Sie die Schaltfläche **Neu**. Sie werden aufgefordert, einen Namen für den neuen Modul zu vergeben. Als Ergebnis sehen Sie den neuen Modul als Unterpunkt der gewählten Bibliothek. Schließen Sie den Dialog mit **Schließen**.

Im weiterhin geöffneten darunterliegenden Dialog, aus dem wir den gerade geschlossenen Dialog aufgerufen hatten, doppelklicken Sie ggf. die Bibliothek oder klicken Sie auf das Plus-Zeichen vor dem Namen, um den Unterpunkt anzuzeigen. Dann doppelklicken Sie den Modul, um den Inhalt dieses Moduls anzuzeigen. Das ungefähre Ergebnis zeigt Abbildung 4.



Abbildung 4: 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 (s. „Die Basic-IDE“ auf Seite 19). Das Makro mit dem Namen „Main“ können Sie bedenkenlos überschreiben, es dient nur als Platzhalter für von Ihnen zu erstellenden Code.

## Makros aufzeichnen

### 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.

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 Dispatcher-Befehl umgesetzt).

Der Makrorecorder speichert eine Abfolge dieser Dispatcher-Befehle 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 späteren Abschnitten vorgestellt werden, primär direkt auf den Dokumenten und greifen nur bei Bedarf auch auf Informationen der Benutzeroberfläche zu.

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 die Eigenschaften des Objekts 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.

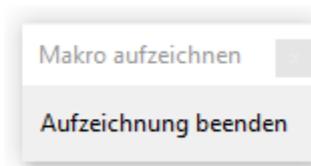
Der Makrorecorder bietet aber denjenigen, die sich nicht mit der Programmierung befassen möchten, in manchen Fällen die Möglichkeit, trotzdem Programme zu erzeugen, die das Arbeiten erleichtern.

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

Als ein erstes Beispiel zeichnen wir ein Makro auf, das einen Text an der Stelle des Cursors in ein Dokument einfügt. Das kann z.B. nützlich sein, wenn Sie den gleichen Text häufiger einfügen wollen, aber zwischendrin auch andere Aktionen vornehmen, die eventuell die Zwischenablage überschreiben, so dass Sie dann den wiederholt einzufügenden Text wieder erneut in die Zwischenablage kopieren müssten.

- Positionieren Sie ggf. den Cursor dort, wo Sie den gewünschten Text zum ersten Mal einfügen möchten, oder wo Sie ihn nach Erstellung des Makros bequem wieder löschen können.
- 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 5).



*Abbildung 5: 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 an die Stelle im Dokument setzen, mit der Sie das Makro aufzeichnen wollen.
- Tippen Sie den Text ein, z.B. „LibreOffice“.
- Beenden Sie die Makroaufzeichnung durch Anklicken des entsprechenden Befehls im schwebenden Fenster.
- Es wird nun ein Fenster geöffnet, in dem Sie das Makro speichern können (Abbildung 6).

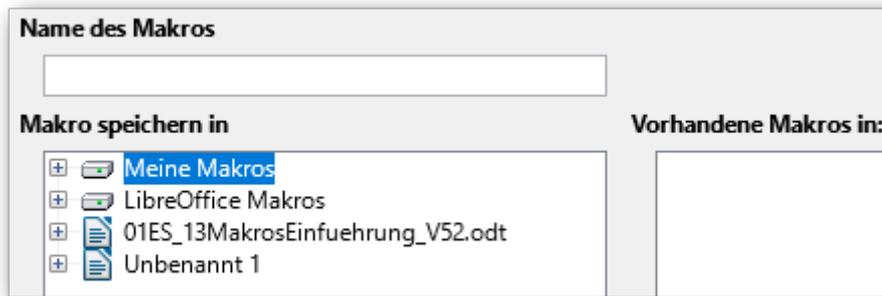


Abbildung 6: Speichern eines aufgezeichneten Makros

- Je nachdem, wo Sie das Makro speichern wollen (s. „Wo sollen Makros gespeichert werden?“ auf Seite 9), wählen Sie nun den Ort der Speicherung. Achten Sie insbesondere, wenn Sie in „Meine Makros und Dialoge“ speichern wollen, darauf, dass Sie die richtige Bibliothek und den richtigen Modul gewählt haben, denn sonst müssen Sie später mühsam den Ort suchen, an dem das Makro gespeichert ist. Ggf. erzeugen Sie vor der Aufzeichnung des Makros eine neue Bibliothek und/oder einen neuen Modul.
- Für unser Beispiel wählen wir die Speicherung im Dokument und dementsprechend den Speicherort „Unbenannt 1“ (es ist für die Testzwecke ein neues, eventuell noch gar nicht gespeichertes Dokument) und **Speichern**. Das Programm schlägt nun den Modulnamen „Module 1“ vor, den wir akzeptieren, und zeigt als Ergebnis:

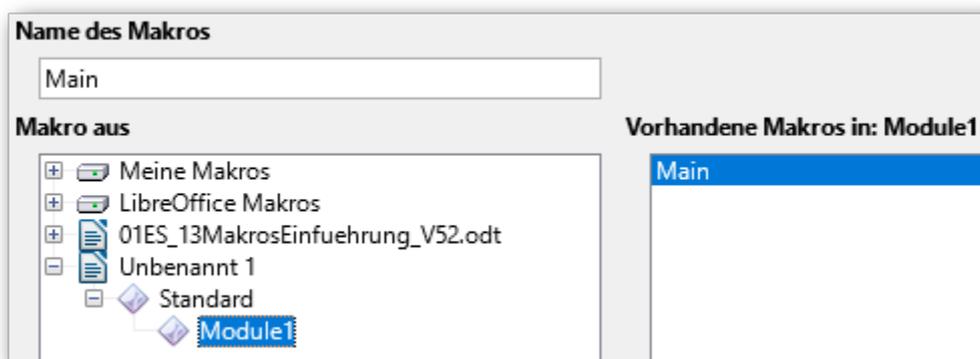


Abbildung 7: Einfügen neues Makro in einem Dokument

Als Bibliothek wurde die Standardbibliothek gewählt, das Makro erhält, wie schon in „Erstellen eines Moduls“ auf Seite 11 beschrieben, den provisorischen Namen „Main“, den Sie ändern sollten, wenn Sie das Makro nicht nur kurzfristig beim Erstellen des Dokuments benötigen.

### Das aufgezeichnete Makro

Das aufgezeichnete Makro sieht folgendermaßen aus. Der Code ist ungefähr so eingefärbt, wie Sie ihn beim Verwalten sehen (Schlüsselwörter blau, Variable und Funktionen grün, Konstanten rot und Kommentare der besseren Lesbarkeit halber schwarz statt wie im Original grau).

```

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 -----
dim args1(0) as new com.sun.star.beans.PropertyValue
args1(0).Name = "Text"
args1(0).Value = "LibreOffice"

dispatcher.executeDispatch(document, ".uno:InsertText", "", 0, args1())

end sub

```

Wir wollen das Makro hier nicht im Einzelnen erläutern, wenn Sie sich mit Basic beschäftigen wollen, werden Sie ohnehin auch den diesbezüglichen Abschnitt „Erstellen von Makros“ ab Seite 19 dieses Kapitels lesen, in dem Sie die Erklärungen finden.

Wir möchten nur auf das hinweisen, was spezifisch für aufgezeichnete Makros ist:

- Mit `dispatcher = createUnoService("com.sun.star.frame.DispatchHelper")` wird ein sogenannter Service erzeugt, der dann aufgerufen werden kann, um die gewünschte Aktion durchzuführen.
- `dispatcher.executeDispatch(document, ".uno:InsertText", "", 0, args1())`: hiermit führt der Dispatcher den angegebenen Befehl, hier das Einfügen von Text, aus. Die Informationen für diesen Befehl werden in der Variablen `args1` mitgegeben, die aus einem Paar von Angaben besteht, wobei die erste den Typ beschreibt und die zweite den Inhalt.

Alle Makros, die aufgezeichnet werden, verwenden diesen einen Dispatcher-Aufruf, nur mit unterschiedlichen Befehlen in der Klammer und mit anderen Argumenten, wobei das auch mehrere Paare sein können (s. „Ein komplizierteres Beispiel“ auf Seite 16).

### **Schnelleres Ausführen des Makros**

Die Ausführung des Makros wäre auf dem in „Makro direkt ausführen“ auf Seite 7 beschriebenen Weg, bei dem Sie immer wieder den Menüpunkt aufrufen und dann das Makro wählen müssen, wahrscheinlich nicht effektiver als das erneute Kopieren des einzufügenden Texts von der letzten Stelle in die Zwischenablage. Aber wie in „Makro über die Makroverwaltung ausführen“ auf Seite 7 bereits angedeutet, lässt sich das wiederholte Ausführen des gleichen Makros beschleunigen:

1. Rufen Sie **Extras** → **Makros** → **Makros bearbeiten** auf
2. Wählen Sie das gerade gespeicherte Makro aus und doppelklicken Sie es.
3. Im rechten Teil des Fensters wird dann das Makro angezeigt. Abbildung 8 zeigt rechts nur einen kleinen Ausschnitt davon.
4. Betätigen Sie das Symbol für das Ausführen des Basic-Programms (in der Abbildung eingekreist)
5. Für weitere Aufrufe müssen Sie nur wieder in dieses Fenster wechseln und das Symbol betätigen.

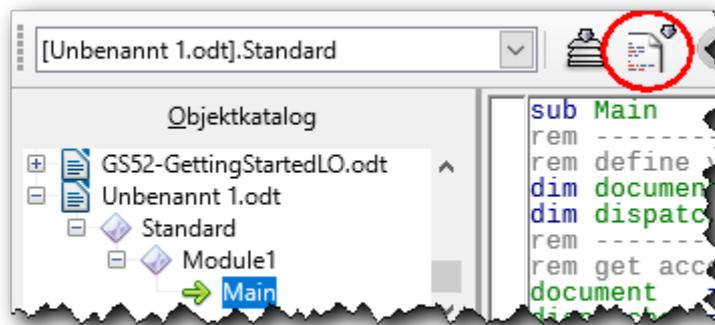


Abbildung 8: Aufruf eines Makros aus der Basic-IDE

## Ein komplizierteres Beispiel

Um ein Beispiel für eine Aufzeichnung zu zeigen, das sich nicht ohne weiteres durch andere Standard-Befehle lösen lässt, verwenden wir das Thema der IBAN (s. „Beispiel 1: IBAN berechnen“ auf Seite 21, das auch für die Beispiele bei der Makroprogrammierung benutzt wird: 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.

Um dieses Makro aufzuzeichnen, gehen Sie wie folgt vor:

1. 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.
2. Wählen Sie wieder **Extras** → **Makros** → **Makro aufzeichnen**.
3. Um nun ein Makro aufzuzeichnen, das unabhängig davon funktioniert, wo der Cursor im Wort steht, bedienen wir uns der Tatsache, dass es eine Tastenkombinationen gibt, die den Cursor zum Anfang des Wortes bewegt: *Strg+Nach links*. Weil dieser Befehl aber, falls der Cursor schon am Beginn des Wortes steht, diesen zum vorigen Wort bewegt, verwenden wir zusätzlich die Kombination *Strg+Nach rechts*, um zunächst zum Beginn des nächsten Wortes zu springen; wenn wir dann *Strg+Nach links* auslösen, springt der Cursor zum Beginn des nun vorigen Wortes, also genau zu dem Beginn des Wortes, das wir in Vierergruppen aufteilen wollen. Betätigen Sie also nacheinander *Strg+Nach rechts* und *Strg+Nach links*.
4. Betätigen Sie nun viermal die Taste *Nach rechts* und danach die *Leertaste*: nach vier Zeichen ist damit ein Leerzeichen eingefügt.
5. Wiederholen Sie das noch viermal: Sie haben dann die IBAN in Vierergruppen aufgeteilt, zuletzt stehen zwei Zeichen.
6. Beenden Sie die Makroaufzeichnung durch Anklicken des Befehls im schwebenden Fenster
7. Speichern Sie das Makro.

## Das aufgezeichnete Makro

Von dem aufgezeichneten, ziemlich langen Makro zeigen wir nur einen kleinen Teil, um noch einige Erläuterungen hinzuzufügen (die Sie übergehen können, wenn Sie das Makro nur speichern wollen).

```
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:GoToNextWord", "", 0, Array())

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

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 -----
dim args4(1) as new com.sun.star.beans.PropertyValue
args4(0).Name = "Count"
args4(0).Value = 1
args4(1).Name = "Select"
args4(1).Value = false

dispatcher.executeDispatch(document, ".uno:GoRight", "", 0, args4())

```

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

```

rem -----
dim args7(0) as new com.sun.star.beans.PropertyValue
args7(0).Name = "Text"
args7(0).Value = " "

dispatcher.executeDispatch(document, ".uno:InsertText", "", 0, args7())

```

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 wie der hier abgedruckte!

Sie sehen, dass die Argumente für den Dispatcher-Aufruf nun beim Gehen nach rechts aus zwei Paaren bestehen, nummeriert von 0 bis 1. Beachten Sie auch, 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.

In „Beispiel 5: Ersatz des aufgezeichneten IBAN-Makros durch ein programmiertes“ auf Seite 38 zeigen wir, wie die gleiche Aufgabe durch Programmierung gelöst werden kann.

# Programmieren von Makros mit Basic

---

## Hinweise bzgl. der 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. Sie werden dadurch mit den am häufigsten benötigten Befehlen bekannt werden.

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 verketteten, 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, 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 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, 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 31) 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 40 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.

## Makros bearbeiten

Bisher haben wir noch keinen Makro-Code selbst eingegeben, da das durch die Aufzeichnung mit erledigt wurde. Bei selbsterstellten (oder von woanders kopierten) Makros muss der Code jedoch zunächst einmal in LibreOffice eingegeben und gespeichert werden. Wenn Sie eines der Beispiel-Makros ausprobieren wollen, können Sie es einfach in die Zwischenablage kopieren und in der Basic-IDE, die nun beschrieben wird, einfügen.

## Die Basic-IDE

Die IDE (Integrated Development Environment = Integrierte Entwicklungsumgebung) ist das Werkzeug, mit dem Sie Makros schreiben und ändern (Abbildung 9).

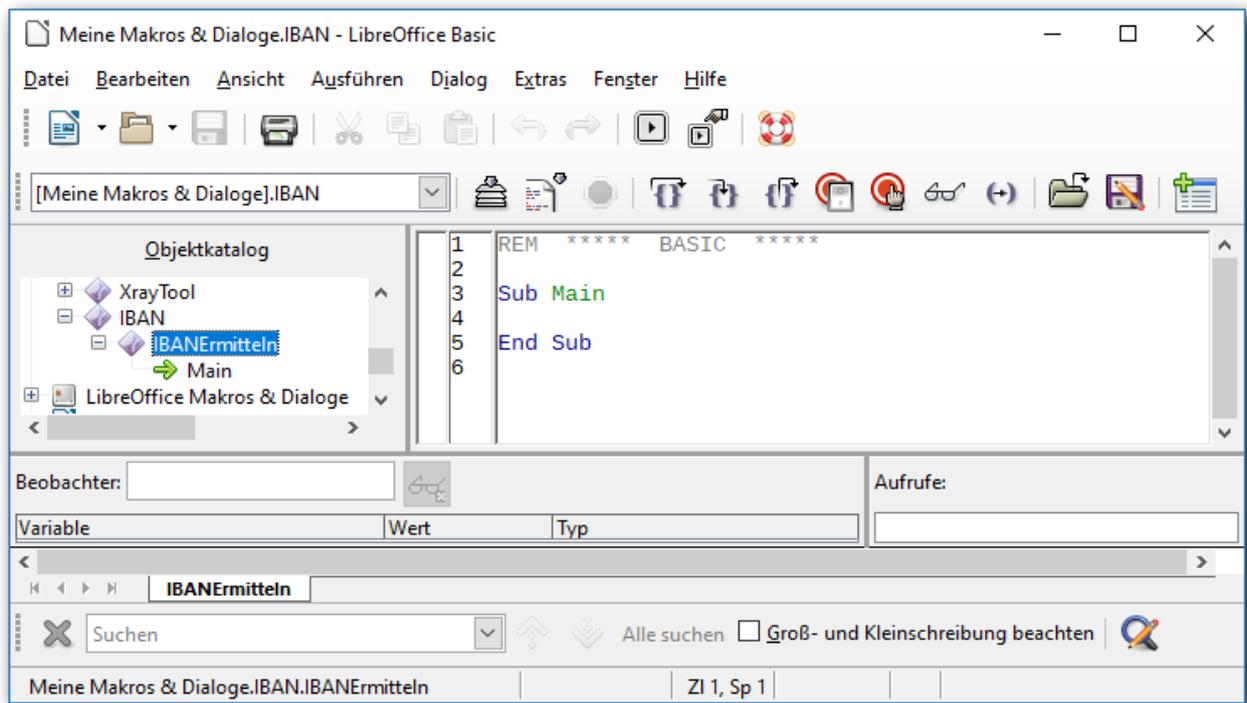


Abbildung 9: die Basic-IDE. Der Lesbarkeit wegen ist das Feld für den Code wesentlich verkleinert

- 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.

Den Code geben Sie im rechten – normalerweise – großen Feld der IDE ein. Wenn Sie einen neuen Modul erstellt haben, enthält er ein Makro namens *Main*, das keine weiteren Befehle enthält. Dass es erzeugt wird, hat sowohl praktische als auch historische Gründe. Sie können den

Namen ändern und ihren Code einfügen oder es löschen. Mit solch einem wenig aussagekräftigen Namen sollte man normalerweise nicht arbeiten.

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.

## 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 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. Sie können das fehlerhafte Programm über die Zwischenablage abspeichern und dann ausführen, wie in „Makros ausführen und speichern“ ab Seite 6 beschrieben, wenn Sie selbst die Reaktion von LibreOffice sehen wollen, der Fehler ist aber im Text so genau dargestellt, dass Sie auch erst mit dem korrigierten Makro Ihren ersten Speicherversuch machen können.

#### Vorsicht



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

```
sub ermiddleIBAN
```

```
REM vereinfachte Variante:
```

REM nur für Deutschland, nur Standardfälle abgedeckt.

REM 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
```

- 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 angestückt.

- 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, denn hier passiert der angekündigte Fehler.
- 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 10)

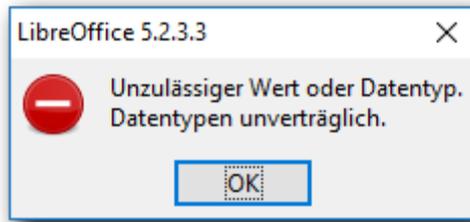


Abbildung 10: Fehlermeldung bei der Ausführung des Makros

und die Zeile, in der der Fehler auftritt, ist markiert (Abbildung 11):

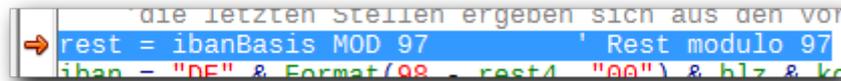


Abbildung 11: markierte Zeile bei Fehler

### 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.

#### Tip

Deklariert Sie immer alle Variablen, dann sind Sie sicher, dass der richtige Typ verwendet wird. Verwenden Sie am besten am Beginn jedes Moduls die Anweisung „Option Explicit“, dann weist Sie LibreOffice darauf hin, dass Variablen nicht deklariert sind.

Wenn wir das so geänderte Makro ausführen, erhalten wir zwar wieder eine Meldung (Abbildung 12), aber die bringt uns weiter:

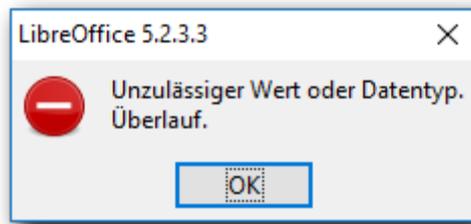


Abbildung 12: Fehlermeldung aus geändertem Makro

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:
    REM nur für Deutschland, nur Standardfälle abgedeckt.
    REM    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 gleich von 1 bis 24 laufen lassen und die anfängliche Zuweisung für *rest* dann weglassen. 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 auch 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 27) verwendet, aber für das Verständnis der Umsetzung der schriftlichen Division war die hier gezeigte Variante zunächst nützlicher.

Wenn dieses Programm ausgeführt wird, erscheint als Ergebnis das folgende Fenster mit der korrekten IBAN (Abbildung 13):



Abbildung 13: 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 40 beschrieben sind.

### 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; eine Prüfung wäre sehr nützlich.
- 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:
REM nur für Deutschland, nur Standardfälle abgedeckt.
REM 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 Beendigung 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 25ff. 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 13), allerdings führen Eingabefehler nun nicht mehr zu eventuellen rätselhaften Abbrüchen, sondern zu verständlichen Meldungen wie in Abbildung 14.

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



Abbildung 14: Fehlermeldung für nichtnumerische Kontonummer

## Makros im produktiven Einsatz

### Beispiel 4: Massensetzung der IBAN in einer Calc-Tabelle

Kontoverbindungen hat man vielleicht 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 15):

	A	B	C
1	konto	BLZ	IBAN
2	123456	74297531	
3	1122334455	76543210	
4			

Abbildung 15: Testtabelle 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 18)

- 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; da es sich nur um ein Testbeispiel handelt, kann das ruhig in der Bibliothek „Standard“ geschehen (s. „Erstellen eines Moduls“ auf Seite 11).

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 16):

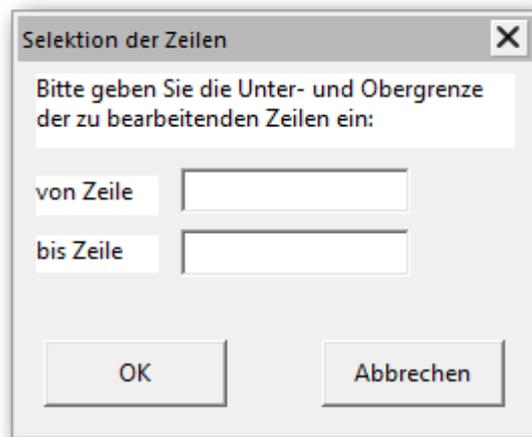


Abbildung 16: Der Dialog für Unter- und Obergrenze

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 „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 17). 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.

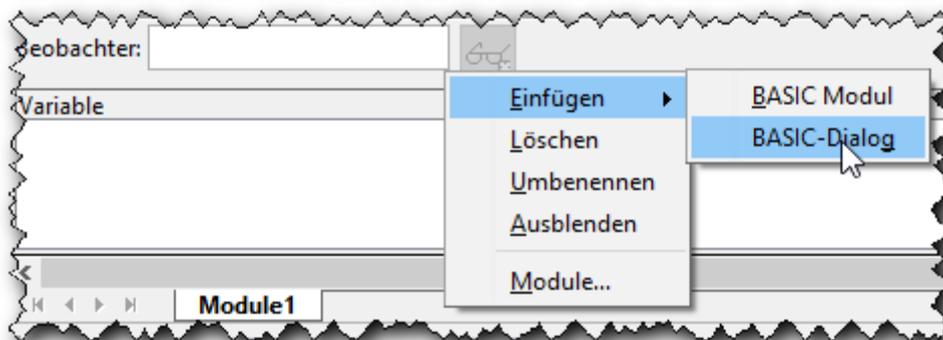


Abbildung 17: Anlegen eines weiteren Moduls oder eines Dialogs

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 beim Lesen des Programms 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 18 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 Fällen auch, ist das erste verwendete Objekt *ThisComponent*. Dieses Objekt ermöglicht den Zugriff auf das aktive Dokument – in unserem Fall ein 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 sich der Cursor befindet, 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 Eigenschaft 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 – im Beispiel *calcDoc* – 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. Wenn wir gerade erst den Dialog erstellt haben und das Programm testen, 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 – angelegten Dialogmusters 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, ohne eine Abfrage der Art „x = y“, 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 27 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 liefern. Nur bei der Übergabe als Zeichenkette wirken unsere Prüfungen.
- Wie in „Beispiel 3: IBAN ermitteln mit Prüfungen, Unterprogrammaufruf“ auf Seite 27 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 *c/IBAN* 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, dann werden wir mit dem von uns eingerichteten Dialog aufgefordert, die Unter- und Obergrenze anzugeben, Abbildung 18 zeigt den Dialog mit bereits eingegebenen Werten.

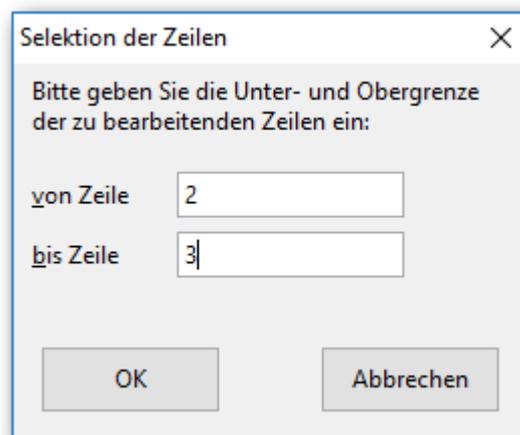


Abbildung 18: Dialog *DlgSel* bei der Ausführung

Wenn Sie *OK* betätigt haben, wird der Rest des Makros durchlaufen, im Beispielfall ist das Ergebnis eine ausgefüllte Tabelle wie in Abbildung 19.

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

Abbildung 19: Calc-Tabelle mit eingefügter IBAN

Wenn allerdings in einer Zeile ein Problem auftritt, dann wird im Unterprogramm *FuncErmittleIBAN* die zugehörige 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 ziemlich 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 40), 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 40 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-Ermittlung

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.

## Aufzeichnen versus Programmieren

Wem es ausreicht, ein 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 meist 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 IBAN-Makros durch ein programmiertes

Wenn man das aufgezeichnete Programm für die IBAN (s. „Ein komplizierteres Beispiel“ auf Seite 16) genauer anschaut, sieht man, wie schon dort angedeutet, dass der zweite bis vierte Block bis auf die anderen Variablennamen `args3`, `args4` usw. gleich aussehen wie der erste, dass also die vierfache Ausführung des Dispatch-Befehls immer mit dem bereits verwendeten `arg3` 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, einer Schleife und des Bewegens um vier Stellen auf einmal, was das Programm wesentlich knapper und auch besser lesbar macht:

```
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. (Still-schweigend 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 Funktio-

nen, 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 allerdings 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 sinnvoll 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 32 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 41). 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 ererbt 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 41) in sehr nützlicher und anwenderfreundlicher Form.

## Bücher

Das umfassende deutschsprachige Buch über die Programmierung mit Makros von Thomas Krumbein mit dem Titel „Makros-Grundlagen“, der auch ein aktiver Teilnehmer auf den Diskussionsseiten von LibreOffice sowie ein aktives Mitglied in der Document Foundation ist, die das Projekt LibreOffice trägt, ist seit kurzem in aktualisierter Form wieder verfügbar:

<http://www.tintal.de/index.php/buecher/fbuecher/31745-makro-grundlagen>.

Wer Französisch versteht, kann auf das exzellente Buch von Bernard Marcellly & Laurent Godard, Programmation OpenOffice.org et LibreOffice zurückgreifen:

<https://eyrolles.com/Informatique/Livre/programmation-openoffice-org-et-libreoffice-9782212132472>. Bei Basic hat sich praktisch nichts verändert, daher ist dieser Stand nach wie vor

äußerst nützlich.

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 wesentlich 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 hinzugefügt, und mittels eines Funktionsaufrufs in einem Programm oder auch – bei vorheriger entsprechender Einrichtung durch Sie – unabhängig von einem Programm durch einen Aufruf Ihrerseits ausgeführt. Nach dem Aufruf werden zu dem jeweils ausgewählten Objekt alle Informationen über „Eigenschaften“ – auch komplexe, mit dem Objekt verknüpfte Objekte wie (oben schon einmal genannt) *sheets* zählen dazu – und Methoden usw. angezeigt. Sie können dann durch Doppelklick wieder zu Unterobjekten navigieren usw.

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 Marcellly, 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](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.